

# Erkennung einer Sieben-Segment-Anzeige mit Maschinellem Lernen

Bachelor Arbeit

von

Frau Laura Luise Becker

im Studiengang Elektrotechnik/Informationstechnik

an der HAWK Hochschule für angewandte Wissenschaft und Kunst

Hildesheim / Holzminden / Göttingen

Fakultät Ingenieurwissenschaften und Gesundheit in Göttingen



in Kooperation mit der Firma LaVision GmbH

Erstprüfer: Prof. Dr. rer. nat. Roman Grothausmann

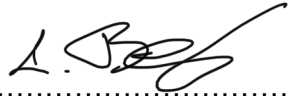
Zweitprüfer: Jens Gutzeit

27. September 2021

## Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Abschlussarbeit selbstständig, ohne fremde Hilfe und nur unter Verwendung der angegebenen Literatur angefertigt habe. Alle fremden, öffentlichen Quellen sind als solche kenntlich gemacht. Mir ist bekannt, dass ich für die Quellen Dritter in dieser Arbeit die Nutzungsrechte zur Verwendung in dieser Arbeit benötige. Weiterhin versichere ich, dass diese Abschlussarbeit noch keiner anderen Prüfungskommission vorgelegen hat.

Göttingen, den 27.06.2021

Unterschrift 


## Erklärung zu Nutzungsrechten und Verwertungsrechten<sup>1</sup>

Ich bin hiermit einverstanden,

dass von meiner Abschlussarbeit (ggf. nach Ablauf der Sperre) 1 Vervielfältigungsstück erstellt werden kann, um es der Bibliothek der HAWK zur Verfügung zu stellen und Dritten öffentlich zugänglich zu machen.

Ich erkläre, dass Rechte Dritter der Veröffentlichung nicht entgegenstehen.

Göttingen, den 27.06.2021

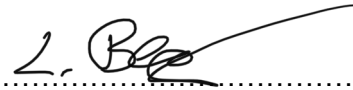
Unterschrift 

## Sperrvermerk der Abschlussarbeit

☒ NEIN\*

JA / Dauer der Sperre: [ ] 3 Jahre\* [ ] 5 Jahre\*

Göttingen, den 27.06.2021

Unterschrift 

\* Zutreffendes bitte ankreuzen

<sup>1</sup> Dadurch räumen Sie der HAWK ein einfaches, zeitlich unbeschränktes, unentgeltliches Nutzungsrecht nach §§ 15 Abs. 2 Nr. 2, 16, 17, 19a, 31 Abs. 2 UrhG ein.

# Inhaltsverzeichnis

|   |           |
|---|-----------|
| <b>Inhaltsverzeichnis</b>                           | <b>1</b>  |
| <b>1 Einleitung</b>                                 | <b>3</b>  |
| 1.1 Gegenstand und Ziel                             | 4         |
| 1.2 Aufbau der Arbeit                               | 4         |
| <b>2 Grundlagen</b>                                 | <b>6</b>  |
| 2.1 Maschinelles Lernen                             | 6         |
| 2.2 Arbeitslauf                                     | 7         |
| 2.3 Überwachtes Lernen                              | 8         |
| 2.4 Neuronale Netzwerke                             | 11        |
| <b>3 Verwendete Technik und Programmiersprachen</b> | <b>17</b> |
| 3.1 Python  | 17        |
| 3.2 DaVis   | 18        |
| 3.3 Google Colab                                    | 19        |
| 3.4 C++   | 19        |
| <b>4 Daten</b>                                      | <b>20</b> |
| 4.1 Aufnahmen                                       | 21        |
| 4.2 Bearbeitung                                     | 22        |
| 4.3 Datenformat                                     | 25        |
| <b>5 Modell</b>                                     | <b>27</b> |
| 5.1 Neuronales Netzwerk                             | 27        |
| <b>6 Diskussion</b>                                 | <b>32</b> |
| 6.1 Ergebnis und Problem des ML                     | 32        |
| 6.2 Vergleich der Methoden                          | 33        |
| <b>7 Ausblick</b>                                   | <b>36</b> |
| 7.1 DaVis   | 37        |
| <b>8 Zusammenfassung und Fazit</b>                  | <b>38</b> |

|  |    |
|--|----|
| <b>Danksagung</b> . . . . .            | 40 |
| <b>Literaturverzeichnis</b> . . . . .  | 41 |
| <b>Abbildungsverzeichnis</b> . . . . . | 43 |

# 1 Einleitung

In der sich immer schneller verändernden Welt von Software und Hardware ist Qualitätssicherung eine große Aufgabe. Die Zyklen zwischen Updates und neuen Treibern werden stetig kleiner und die eingebauten Features immer komplexer. Das macht eine Qualitätssicherung unabdingbar, aber auch extrem aufwendig. Das Automatisieren dieser Aufgabe ist ein großes Bestreben in der Industrie.

So werden Zeit und Ressourcen gespart, zusätzlich sind Maschinen meistens gründlicher in der Ausführung von wiederholenden Tätigkeiten. Das automatische Testen kann nach den Arbeitszeiten passieren und so nochmals Zeit sparen.

Die Software, welche die Automatisierung übernimmt, beruht in jüngster Zeit immer mehr auf Maschinellern Lernen. Dieses bietet sich gerade an, wenn es schon große Datenmengen gibt.

In dieser Arbeit soll die Nützlichkeit und Umsetzbarkeit des Maschinellen Lernens an einem Teil eines schon vorhandenen Teststandes geprüft werden. In dem davor bearbeiteten Praxisprojekt wurde die automatische Erkennung des Teststandes mit konventionellen Programmiermethoden durchgeführt.

Es handelt sich um den Highspeed-Kamera-Teststand der Firma LaVision GmbH. LaVision GmbH aus Göttingen (gegründet 1989) vertreibt Hard- und Software im Bereich der optischen Messtechnik für industrielle und wissenschaftliche Anwendungen. [Lav] Zu der Hardware gehören unter anderem Kameras (High- und Lowspeed), Laser und PCs. DaVis, die von LaVision entwickelte Software, ist eine Komplettlösung für intelligente Imaging-Applikationen, steuert die Hardware an, macht die Aufnahmen, die Bildverarbeitung und Bildauswertung. DaVis wird in C++ implementiert. Bei optischen Messungen werden viele Geräte mit einer extrem genauen zeitlichen Steuerung benutzt, dafür verwendet LaVision eine eigene Hardware Programmed Timing Unit (PTU). Diese steuert den richtigen zeitlichen Ablauf der Aufnahmen indem das Auslösesignal (Trigger) mit der Aufnahmesoftware (DaVis) synchronisiert wird.

## 1.1 Gegenstand und Ziel

Der Highspeed-Kamera-Teststand ist zur Qualitätssicherung von neuen Kameras aber auch der internen Software gebaut worden. Die Teststand-Hardware besteht aus einer digitalen Uhr, welche aus zwei Reihen mit jeweils vier Sieben-Segment-Anzeigen besteht, und zwei Programmed Timing Units, wobei eine die Uhr ansteuert und die andere den gesamten Ablauf der Aufnahme steuert. [Bec21]

In dem Praxisprojekt zur dieser Arbeit wurde eine automatische Erkennung der Sieben-Segment-Anzeige programmiert. Hierzu wurden die einzelnen Segmente angesteuert und die Position und die Helligkeitswerte abgespeichert. Dann wurde bei einer Aufnahme verglichen, ob an der Position der Segmente die Helligkeitswerte erreicht wurden. Aus den erkannten Segmenten wird dann die Zahl erschlossen.

In dieser Arbeit soll nun eine Sieben-Segment-Anzeige mit Maschinellern Lernen erkannt werden. Es soll sowohl das Ergebnis als auch der Aufwand verglichen werden. Hierzu ist zu sagen, dass die Software DaVis der Firma LaVision bisher keine Möglichkeit für Maschinelles Lernen bietet. Aber es gibt sowohl schon eine Python-Bibliothek, um mit Daten aus der Software umzugehen, als auch ist DaVis darauf ausgelegt, viele Aufnahmen zu nehmen und damit weiter zu arbeiten.

## 1.2 Aufbau der Arbeit

Zum Beginn der Arbeit wurde sich als erstes in Materie und mathematische Grundlagen des Maschinellen Lernens eingearbeitet. Die Erkenntnisse werden in Kapitel 2 Grundlagen vorgestellt.

Bei dem weiteren Verlauf wurde sich an den allgemeinen Ablauf von Maschinellen Lernen-Projekten gehalten[Gé19]:

- Problemidentifizierung
- Datenbeschaffung
- Datenanalyse und Visualisieren der Daten
- Datenvorbereitung
- Modellwahl

- Evaluation und Verbesserung des Modelles
- Ergebnis diskutieren

Es wurde die Entscheidung getroffen, nur mit einer Ziffer ein Maschinelles-Lernen-Modell zu implementieren. Der Highspeed-Kamera-Teststand besteht eigentlich aus acht Ziffern in zwei Reihen mit vier Ziffern unterteilt. Der Grund für diese Entscheidung war, dass die Zeit für Datenaufnahmen nicht gereicht hätte. Außerdem ist auch die Frage wie sinnvoll es ist, mit allen Ziffern ein Modell auszustellen oder statt dieses Modells andere Methoden zu verwenden. Dieses wird kurz in dem Kapitel Ausblick besprochen.

In dem Kapitel 3 werden überblicksartig die verwendete Technik und die Programmiersprachen vorgestellt. In der Arbeit wurde vor allem Python benutzt. Da aber in der Firma LaVision C++ die Hauptprogrammiersprache ist, wird in Kapitel 6 auf die Möglichkeiten von Maschinellen Lernen in dieser Sprache eingegangen.

In dem Kapitel 4 und 5 wird der größte Teil des Maschinellen Lernen-Projekts umgesetzt. In Kapitel 4 wird sich mit den Daten auseinander gesetzt. Im Kapitel 5 werden dann unterschiedliche Modelle aufgebaut und verbessert. Zum Schluss gibt es im Kapitel 7 eine Diskussion über die Ergebnisse des Maschinellen Lernens und einen Vergleich zu der konventionell programmierten Methode.

## 2 Grundlagen

In diesem Kapitel wird der mathematische Hintergrund des Maschinellen Lernens beschrieben. Zur Veranschaulichung wird die Problemstellung der Bachelorarbeit benutzt, bei der es darum geht, Ziffern einer Sieben-Segment-Anzeige zu erkennen.

### 2.1 Maschinelles Lernen

Beim Maschinellen Lernen geht es darum, dass der Computer aus zur Verfügung gestellten Datensätzen selbstständig Muster und Regeln erkennt. [Mat21]

Es wird also kein Programm mehr geschrieben, mit dem in unserem Beispiel genau beschrieben werden muss, wie genau die verschiedenen Ziffern erkannt und differenziert von einander werden. Stattdessen wird ein Algorithmus geschrieben, der an Hand von Daten sich selbstständig aktualisiert und lernt. Dies wird als Modelltraining bezeichnet. Mit dem abgeschlossenen Modell können Vorhersagen über neue Daten getroffen werden. So kann das Modell zur Ziffernerkennung aus einem Bild einer Eins bestimmen, mit welcher Wahrscheinlichkeit es sich bei der Ziffer um eine Eins handelt.

Es gibt drei Hauptrichtungen des Maschinellen Lernens: Überwachtes Lernen, Unüberwachtes Lernen und Verstärktes Lernen. Bei dem Überwachten Lernen wird mit einem bekannten Datensatz und auch vorgebenen Zielvariablen gelernt. Es soll der Zusammenhang zwischen den beiden Werten gelernt werden und richtig prognostiziert werden. Diese Art von Maschinellern Lernen wird in dieser Arbeit verwendet.

Beim Unüberwachten Lernen bekommt der Algorithmus Daten, aber keine Zielvariablen und muss die Daten selbst in Gruppen oder Muster einteilen.

Beim Verstärkten Lernen interagiert der Algorithmus mit der Umgebung und es gibt ein Belohnungssystem. So entwickelt der Algorithmus selbstständig Lösungen zu einem Problem.[Urla]

Im weiteren wird sich nur noch auf das Überwachte Lernen bezogen.



## 2.2 Arbeitslauf

In einem Projekt zum Maschinellen Lernen (ML) wird sich damit beschäftigt, welche Merkmale der Daten wichtig für die Fragestellung sind. In dieser Bachelorarbeit soll eine Sieben-Segment-Anzeige erkannt werden. Dazu müssen möglichst viele verschiedene Daten von dieser Anzeige erhoben und mit einem Label versehen werden. Das Label ist die Zielvariable, nach der Algorithmus unterscheiden soll, in diesem Fall nach den Ziffern Null bis Neun. Nachdem Erheben der Daten werden diese visualisiert und analysiert.

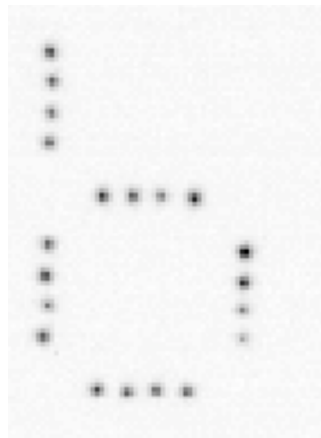


Abbildung 1: Aufnahme mit der Sieben-Segment-Anzeige mit Label: 6

Im nächsten Schritt werden die Daten für den ML Algorithmus vorbereitet, dazu gehört auch die Aufteilung der Daten und bereinigt. Die Unterteilung erfolgt in drei Stücke, sogenannte Sets.

Die Daten werden in Trainings-, Test- und Validierungssets getrennt. Der Algorithmus wird anhand der Trainingsdaten trainiert. Dieses Set sollte ungefähr 70 % des gesamten Datensatzes beinhalten mit gleicher Verteilung von allen Zielvariablen. [Gé19]

Das Testset wird benutzt, um die Qualität des Modells zu beurteilen. Dieses Set sollte ungefähr 20 % des Datensatzes beinhalten. Hierbei ist es wichtig, dass der Algorithmus die Daten nicht zum Lernen benutzt.

Mit dem Validierungsset, welches 10 % der Daten enthält, werden später die Hyperparameter (z.B. die Lernrate oder das Gewicht) des Modells angepasst. Zusätzlich wird es verwendet, um eine Überanpassung (engl.: Overfitting) zu verhindern. Beim Overfitting lernt der Algorithmus des Modells zu viele der spezifischen Kleinigkeiten des Trainingssets und ist nicht mehr so gut beim Generalisieren. Dieses fällt normalerweise

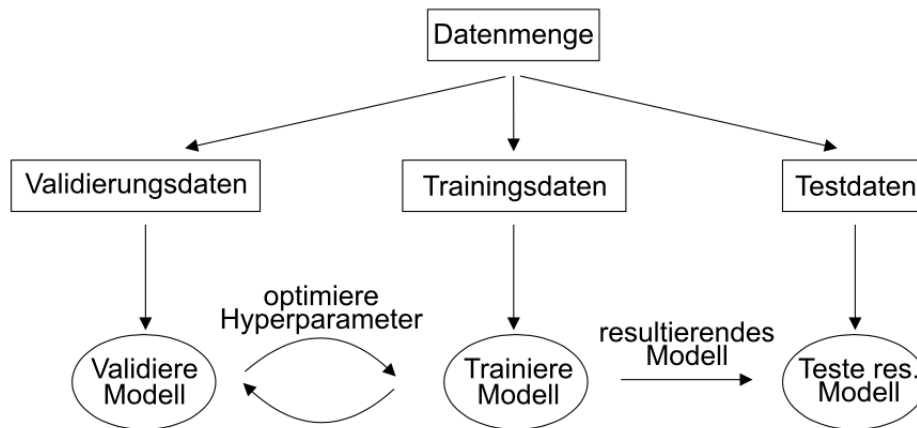


Abbildung 2: Arbeitsablauf Maschinelles Lernen  
[Cho+20]

bei dem Evaluieren des Modelles mit dem Validierungsset auf. Das Validierungsset wird beim Lernen gar nicht verwendet, dadurch kann der Algorithmus nicht seine besonderen Merkmale trainieren. Zusätzlich ist die Erkennungsrate gering, wenn der Algorithmus sich zu sehr spezialisiert hat.

## 2.3 Überwachtes Lernen

Der generelle Gedanke hinter dem Überwachten Lernen ist es, mit den vorhandenen Eingabedaten und Zielvariablen eine Schätzfunktion aufzustellen. Diese Schätzfunktion mit Parametern untersucht wie von den Eingabedaten auf die Zielvariablen geschlossen werden kann. Die Parameter der Schätzfunktion werden in dem Lernprozess des Algorithmus anpasst um die Schätzfunktion zu verbessern. Dazu wird eine Verlustfunktion aufgestellt. Diese quantifiziert die Güte der Schätzfunktion. Also wird beschrieben, wie groß die Abweichung der Schätzfunktion und der Zielvariable ist. Zur Optimierung der Schätzfunktion müssen ihre Parameter angepasst werden, dazu wird die Minimierung der Verlustfunktion benutzt.

Im Überwachten Lernen ist zwischen zwei Arten von Aufgaben zu unterscheiden: der Klassifikation und der Regression. Die Regression versucht aus vorhanden Daten eine Vorhersage oder einen Rückschluss zu treffen. Ein Beispiel wäre, aus Wetterdaten das morgige Wetter vorherzusagen.

Bei der Klassifikation gibt es diskrete Zielvariable und es soll ein neues Eingabeereignis in die Klassifikationskategorien einsortiert werden, z.B das Erkennen von Straßenschildern. Dieses ist auch das Ziel dieser Arbeit. So sollen in einem neuen Eingabebild die Ziffer erkannt werden.

Es gibt verschieden Methoden und Algorithmen die bei dem Überwachten Lernen verwendet werden. Die einfachste ist eine lineare Methode. Diese wird hier einmal vorgestellt, um die mathematischen Hintergründe zu erläutern.

### 2.3.1 Lineare Klassifikation

Bei der Linearen Klassifikation liegen Paare von Eingabewerten und Zielvariablen vor. Als Beispiel sind in dieser Arbeit die Eingabewerte ein Vektor, der das Bild beschreibt. Jedes Pixel hat einen Eintrag für den Helligkeitswert von 0-255. Die Zielvariablen haben die Werte der Ziffern von Null bis Neun.

$$(x_1, y_1), \dots, (x_m, y_m)$$

Bei der Klassifikation wird normalerweise eine Kodierung verwendet statt einfach die natürlichen Zahlen zu nutzen, da diese eine Ordnung untereinander besitzen und der Algorithmus diese erlernen könnte.

Eine Art von Kodierung ist die *One-Hot-Kodierung* dabei wird die Zielvariable zum Einheitsvektor in diese Richtung [Cho+20].

$$y \rightarrow e^{(y)} = \begin{bmatrix} e_1^{(y)} \\ \vdots \\ e_y^{(y)} \\ \vdots \\ e_p^{(y)} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \quad (1)$$

Der Vektor der Eingabewerte

$$x^T = (x_1, x_2, \dots, x_n)$$

und der Parameter  $\beta$  bilden eine lineare Schätzfunktion:

$$f(x|\beta) = \beta_0 + \sum_{j=1}^n \beta_j x_j = \tilde{x}^T \quad (2)$$

mit

$$\tilde{x}^T = (1, x_1, \dots, x_n)$$

und

$$\beta = (\beta_0, \dots, \beta_n)$$

Um den Parameter  $\beta$  zu optimieren wird eine Verlustfunktion minimiert.

$$\beta_{opt} = \operatorname{argmin} L(\beta) \quad (3)$$

Das Minimierungsproblem ist in der Mathematik gut und viel erforscht, deswegen gibt es für ML verschiedene Verlustfunktionen. Hier wird die Verlustfunktion der Quadratsumme der Residuen (RSS) beschrieben. RSS wird auch als L2-Verlust oder Gauß-Verlust bezeichnet. Sie gibt einen Abstand zwischen dem Ergebnis der Schätzfunktion und der Zielvariable an.

$$\text{RSS}(\beta) = \sum_{i=1}^m [y_i - f(x_i|\beta)]^2 \quad (4)$$

Für die Einordnung eines neuen Eingabewertes in eine Klasse wird, nach dem das Optimieren der Schätzfunktion abgeschlossen ist, eine Aktivierungsfunktion benutzt. Sie gibt die Wahrscheinlichkeit, dass die Eingabewerte einer Klasse angehören an. Dabei wird das größte Element aus dem Ergebnisvektor der Schätzfunktion gesucht:

$$F(x|\beta) = \operatorname{argmax}_k f_k(x|\beta) \quad (5)$$

## 2.4 Neuronale Netzwerke

Die Grundidee der Neuronalen Netzwerke beruht auf dem Prinzip von biologischen Nervenzellen (Neuronen). Es wird ein Eingangssignal verarbeitet und erst wenn ein Grenzwert überschritten wird, folgt die Weiterleitung eines Impulses.

Bei einem Neuronalen Netzwerk werden die Eingangswerte mit einer Gewichtung belegt und innerhalb des Neurons aufsummiert. Die Summe wird dann gegen einen Grenzwert überprüft und gegebenenfalls weitergeleitet [Mat21].

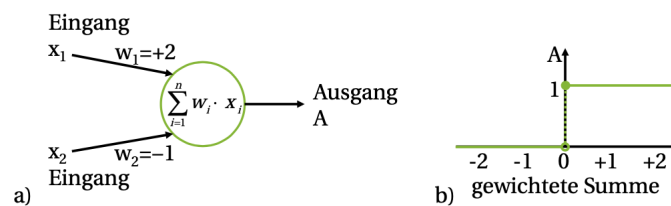


Abbildung 3: Neuron  
[Mat21]

Zur Vereinfachung wird in der Formel der Grenzwert auf die Seite der Eingangswerte ( $x_i$ ) und Gewichte ( $w_i$ ) geholt. Der Grenzwert wird dann als Bias (b) bezeichnet und gehört mit den Gewichten zu den veränderbaren Parameter während des Trainierens des Netzwerkes. [Urlb]

$$\text{Ausgang} = \sum_{i=1}^n x_i w_i + b \quad (6)$$

In einem Neuronalen Netzwerk werden mehrere Neuronen in eine Schicht gebaut und dann mehre Schichten hintereinander verknüpft. Die Schichten zwischen den Eingabe- und Ausgabeschichten werden 'verdeckte Schichten' (engl. hidden Layers) genannt.

In dem Netzwerk werden alle Neuronen einer Schicht mit allen Neuronen in der nächsten Schicht verbunden. Jede dieser Verbindungen hat ihr eigenes Gewicht und die Neuronen haben jeweils eine Bias [Mat21].

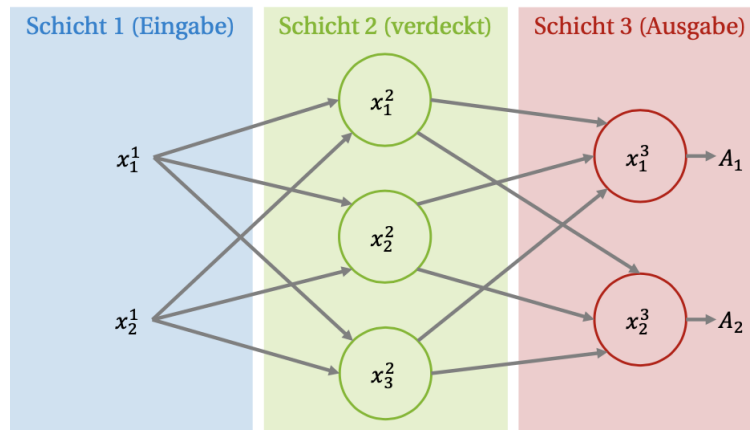


Abbildung 4: Netzwerk Aufbau  
[Mat21]

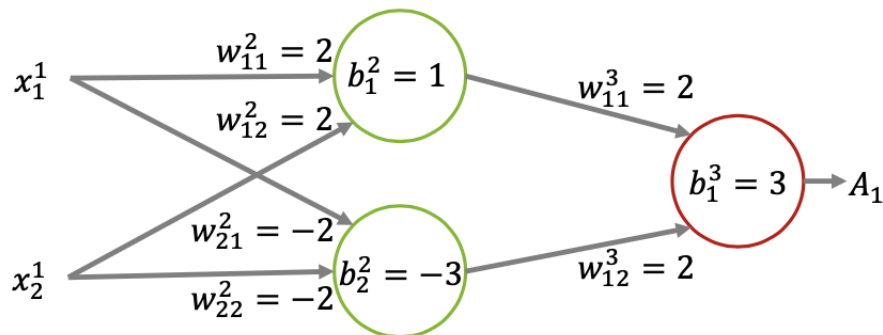


Abbildung 5: Neuronen in mehrschichtigen Netzwerk  
[Mat21]

### 2.4.1 Aktivierungsfunktion

Bei der Gleichung 6 handelt es sich um eine lineare Funktion. Für Neuronale Netzwerke werden nicht-lineare Funktionen als 'Aktivierungsfunktionen' benutzt, da sie besser als lineare Funktionen geeignet sind, unterschiedliche Gewichtungen bei den Eingabewerten abzufangen. Es wird versucht Funktion als Aktivierungsfunktion zu benutzen. Diese haben den Vorteil, dass es beim Lernen, wo differenziert wird, nicht zu Lücken kommt. Außerdem können Funktionen mit begrenzten Wertebereich hilfreich sein. So können zu

große Eingabewerte nicht zu extrem Ergebnissen der Aktiierungsfunktion führen  
Aktivierungsfunktionen [Cho+20]:

**ReLU:** rectified linear unit, alle negativen Zahlen verschwinden, während es eine linear wachsende Funktion für positive Zahlen ist

**Sigmoid:** eine geglättete Version der Stufenfunktion

**Hyberbolische Tangens:** ähnliches Verhalten wie Sigmoid, kann aber für positive und negative Werte annehmen

**Softmax:** oft für die letzte Schicht bei Klassifikationsaufgaben verwendet

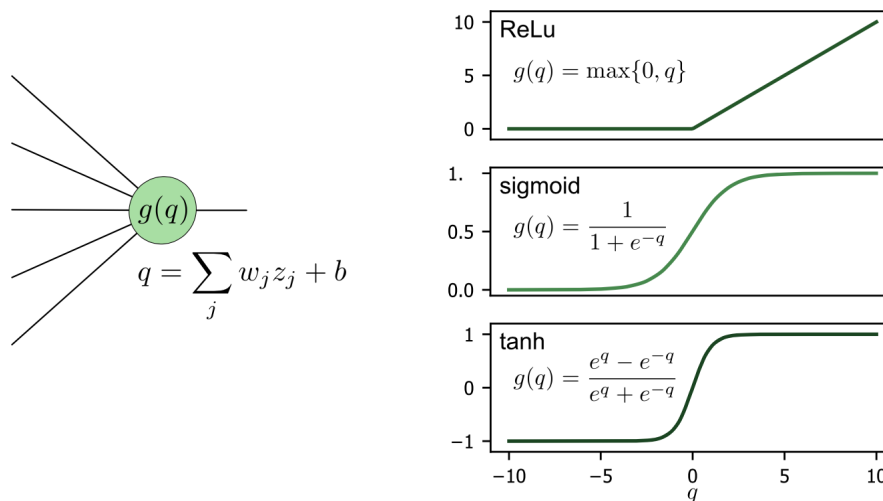


Abbildung 6: Aktivierungsfunktion [Cho+20]

### 2.4.2 Training

Bei dem Training von Neuronalen Netzwerken werden die Parameter Gewichte und Bias optimiert. Hierzu wird auch eine Verlustfunktion benutzt. Diese ist abhängig von den Parameter Gewicht ( $w$ ) und Bias ( $b$ ).

$$L(w, b) = \frac{1}{n} \sum_{i=1}^n \|F(x_i) - z_i\|^2 \quad (7)$$

Dies ist die Verlustfunktion des mittleren absoluten Fehlers mit  $F(x)$  der Schätzfunktion und  $z$  der Zielvariablen. Bei  $\|a\|$  handelt es sich um die Norm, die Länge des Vektors. Eine alternative Verlustfunktion, gerade für Klassifikationsaufgaben, ist die Kreuzentrophie:

$$L_{cross}(w, b) = - \sum_{i=1}^m y_i \cdot \ln(F(x_i)) \quad (8)$$

Das Ziel ist es, die Verlustfunktion zu minimieren, also den Abstand zwischen dem Ergebnis der Schätzfunktion und der Zielvariablen möglichst klein zu bekommen.

### 2.4.3 Gradientenabstieg

In Neuronalen Netzwerken ist die Verlustfunktion normalerweise hochdimensional und kann meistens nicht analytisch gefunden werden. Da numerische Verfahren extrem aufwendig sind, wird meistens nur nach einem lokalen, anstelle eines globalen Minimums gesucht.

Dazu wird der Gradient gebildet und so die Richtung bestimmt, in welche die Parameter  $w$  und  $b$  verändert werden sollen. Diese wird in kleinen Schritten, der Lernrate  $\eta$ , wiederholt.

$$(w_{ij}, b_i) \rightarrow (w_{ij}, b_i) - \eta \frac{\delta L_{w,b}}{\delta(w_{ij}, b_i)} \quad (9)$$



Man kann sich das Ganze vorstellen wie einen Ball, der immer wieder fallen gelassen wird und in die nächste Kuhle rollt. Dabei gibt es die Schwierigkeit, die richtige Größe der Lernrate  $\eta$  zu finden. Bei zu kleiner Lernrate  $\eta$  kommt es nur zu kleinen Veränderungen der Modellparameter  $w$  und  $\beta$  (Abb.7. a). Ist aber die Lernrate  $\eta$  zu groß, so kann das lokale Minimum übersprungen werden und der Algorithmus gerät in eine Schleife (Abb.7 c).

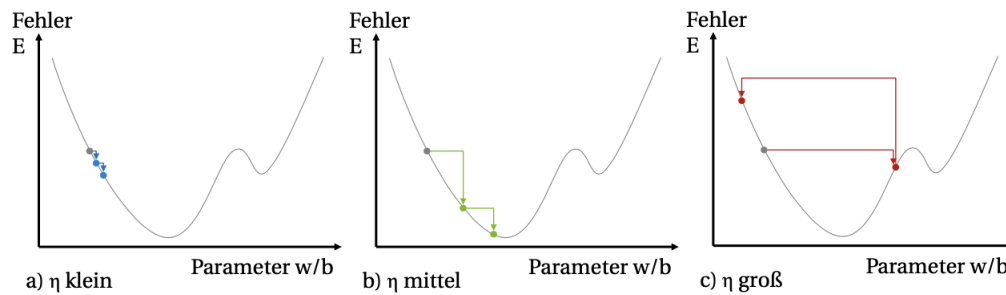


Abbildung 7: Einfluss der Lernrate  
[Mat21]

#### 2.4.4 Aufteilung in Batches

Die Trainingsdaten werden normalerweise nochmals in sogenannte Batches (deutsch: Stapel) unterteilt. Dieses wird getan, damit nicht nach jeden Trainingspunkt die Modellparameter angepasst werden, sondern immer erst, nachdem ein gesamter Batch durchgelaufen ist.

Bei der Batch Unterteilung gibt es drei Hauptvarianten:

**Batch Gradientenabstieg:** Batch Größe = Größe des Trainingssets

**Stochastischer Gradientenabstieg:** Batch Größe = 1

**Mini-Batch Gradientenabstieg:**  $1 < \text{Batch Größe} < \text{Größe des Trainingssets}$

Die Batch Größe ist ein wichtiger Parameter bei Trainieren des Netzwerkes. Auf der einen Seite ist die Rechenzeit, die ein Netzwerk zum Trainieren braucht, desto mehr Berechnung und Gradienten aufgestellt werden, desto länger dauert es. Aber die Wahrscheinlichkeit das global Minimum zu finden steigt natürlich mit der Anzahl der Berechnung.

## 3 Verwendete Technik und Programmiersprachen

Die verwendeten Programmiersprachen und technischen Hilfsmittel werden in diesem Kapitel kurz vorgestellt.

### 3.1 Python

Python ist eine der meist benutzten universellen höheren Programmiersprachen. Zentrales Ziel bei der Entwicklung der Sprache ist die Förderung eines gut lesbaren, knappen Programmierstils. So wird beispielsweise der Code nicht durch geschweifte Klammern (wie in fast allen C-basierenden Sprachen), sondern durch zwingende Einrückungen strukturiert. Zudem ist die gesamte Syntax reduziert und auf Übersichtlichkeit optimiert [Ste18].

Python gehört zu den meist gebrauchten Programmiersprachen für Maschinelles Lernen [Urlc]. Deswegen gibt es viele Bibliotheken, die bei der Datenanalyse, dem Aufbau der Neuronalen Netzwerke und dem Ausführen der Netzwerke helfen. Es ist nicht mehr nötig, einzelne Methoden zum Beispiel für Verlustfunktionen zu programmieren. Diese sind in den Bibliotheken schon enthalten und ausgiebig getestet.

Die benutzten Bibliotheken für Maschinelles Lernen werden einmal kurz vorgestellt. Außerdem wurden noch numpy, benutzt, welches eine der fundamentalen Bibliotheken für wissenschaftliches Arbeiten in Python ist. [Urlc] Mit numpy werden zum Beispiel die Matrizen der Eingabedaten erstellt.

#### 3.1.1 Scikit-Learn

Scikit-Learn ist eine Open-Source-Bibliothek, die sowohl Überwachtes als auch Unüberwachtes Lernen unterstützt. Sie bietet mehrere Methoden für die Datenvorbereitung, Modellselektion und Evaluation [Urle]. In der Arbeit wird Scikit-Learn sowohl zur Datenvorbereitung benutzt als auch zur Unterteilung der Daten in die Trainings-, Test- und Validierungssets. Außerdem wird ein kleines konventionelles Maschinelles Lernmodell damit erstellt und evaluiert.

### 3.1.2 Keras & TensorFlow

Für die Neuronale Netzwerke wurde Keras und TensorFlow benutzt. Keras ist die Schnittstelle für TensorFlow. So ist es einfach und übersichtlich, Neuronale Netzwerke aufzubauen und zu evaluieren. Dieses geschieht z.B. dadurch, dass Keras Bausteine für die Modellbildung zu Verfügung stellt und so das komplexe Thema gut verpackt. Aus diesem Grund ist es für den Anwender deutlich leichter Modelle auf zu bauen, außerdem werden viele Einstellung erstmal automatisch getroffen. TensorFlow stellt die Modelle für die Neuronale Netzwerke zu Verfügung.

### 3.1.3 lvreader

lvreader ist eine kostenfreie Bibliothek von LaVision, die den Zugriff auf DaVis-Daten für Python verfügbar zu machen. Die Schnittstelle kann das DaVis-spezifische Datenformat für Bilder und Vektorfelder lesen und auch wieder zurückschreiben. [Urf]

Mit ihr wurden die aufgenommenen Daten in eine numpy-file umgewandelt, um sie in Python weiter verarbeiten zu können.

## 3.2 DaVis

DaVis ist eine Komplettsoftwarelösung für intelligente Imaging Anwendungen und wurde von der Firma LaVision entwickelt. [Urf].

Die Software wurde in dieser Arbeit genutzt, um die Aufnahmen der digitalen Anzeige zu machen.

### 3.3 Google Colab

Colab ist ein Produkt von Google Research. Mit Colab kann jeder Nutzer beliebigen Python-Code in einem Browser schreiben und ausführen. Colab eignet sich besonders für Maschinelles Lernen, Datenanalyse und den Bildungsbereich. Etwas technischer ausgedrückt, ist Colab ein gehosteter Jupyter-Notebookdienst, der keine Einrichtung erfordert und kostenlosen Zugriff auf Rechenressourcen einschließlich GPUs bietet. [Urli]

Kurz gesagt lässt man mit colab seinen Programm auf einen von Google gestellten Server laufen. Diese Maschinen sind extra für Maschinelles Lernen ausgelegt und haben viele benötigte Python-Bibliotheken (z.B. TensorFlow) vorinstalliert. Es gibt jedem die Möglichkeit sein Modell zu trainieren ohne selbst einen leistungsstarken PC zu haben.

Colab wurde benutzt um das Maschinelle Lernen auszuführen. Die Daten wurden dazu in die Google Cloud hochgeladen. Danach konnte das Jupyter-Notebook benutzt werden, um das Programm für das Maschinelle Lernen zu schreiben.

Jupyter-Notebook ist ein webbasiertes Notizbuch, in dem sowohl Programm als auch Textabschnitte geschrieben und verarbeitet werden können. Die kostenlose Version von Colab kann zwölf Stunden benutzt werden, bevor man sich erneut einloggen muss.

### 3.4 C++

C++ ist eine weitverbreitet objektorientierte Programmiersprache. Sie ermöglicht sowohl die effiziente und maschinennahe Programmierung als auch eine Programmierung auf hohem Abstraktionsniveau. Der Standard definiert auch eine Standardbibliothek, zu der verschiedene Implementierungen existieren. [Urli] In der Firma LaVision wird die eigene Software DaVis in C++ entwickelt.

## 4 Daten

Die Aufnahmen wurden mit dem Highspeed-Kamera-Teststand (Abb.: 8) der Firma LaVision gemacht.

Dazu wurde DaVis (Kap.:3.2) benutzt, um circa 60.000 Aufnahmen von der Sieben-Segment-Anzeige zu machen.

Es wurde sich bei der Anzahl der Aufnahmen an dem MNIST 5 Datensatz orientiert.



Abbildung 8: Highspeed-Kamera-Teststand

Es gibt keine genaue Methode, die Menge der nötigen Daten für das Maschinelle Lernen zu bestimmen, deswegen wird sich oft an schon vorhandenen Beispielen orientiert. Eine andere Variante ist eine repräsentativitätsheuristische Methode, dabei wird versucht abzuschätzen, wie viele Beispiele ein Klasse braucht und dann diese Zahl mit der Anzahl der Klassen multipliziert. Als Beispiel in dieser Arbeit: Zehn Klassen mit zehn verschiedenen Einstellungen à 1.000 Aufnahmen wären insgesamt 100.000 Aufnahmen.

## 4.1 Aufnahmen

Bei den Aufnahmen wurde versucht, möglichst viele Parameter der Aufnahme zu verändern. Bei der Veränderung der Parameter geht es darum, möglichst viel Varianz in die Aufnahmen zu bringen. Einzig der Kamerateyp und das Objektiv (Kamera M-Lite 5M mit 35mm Objektiv) wurden nicht verändert.

Die Parameter, welche verändert wurden:

**Belichtungszeit:** zwischen  $88\ \mu\text{s}$  und  $1000\ \mu\text{s}$  in vier Schritten

**Lichtverhältnisse:** abgedunkelt, Deckenlicht und Deckenlicht mit Tageslicht

**Blende:** zwischen vier und 22 in fünf Schritten

**Abstand:** zwischen 77 cm bis 83 cm in drei cm Schritten

**Neigung:** die Kamera wurde zwischen  $-30^\circ$  und  $+30^\circ$  in sieben Schritten gedreht

**Blickwinkel:** die Kamera hat zwischen einen Winkel von  $-20^\circ$  und  $+20^\circ$  in fünf Schritten auf die Anzeige geschaut

Aus den Bilder wurde dann mit DaVis (Kap.:3.2) das obere rechte Sieben-Segment ausgeschnitten. Dieses ist die Ziffer, die sich bei jeder Aufnahme ändert. Die Aufnahme hat nun eine Größe von  $712 \times 82$  Pixeln.

| Ziffer | Anzahl der Aufnahmen |
|--------|----------------------|
| Null   | 5976                 |
| Eins   | 5976                 |
| Zwei   | 5976                 |
| Drei   | 5976                 |
| Vier   | 5976                 |
| Fünf   | 5976                 |
| Sechs  | 5977                 |
| Sieben | 5976                 |
| Acht   | 5976                 |
| Neun   | 5976                 |

Die Ziffern sind ausgesprochen gleich verteilt in dem Datensatz.

## 4.2 Bearbeitung

Zunächst wurde bei den Daten visuell überprüft, ob die Ziffern noch in der Aufnahme sind und die Ziffern unterschiedlich genug sind. Danach wurden eine Ziffer aus den Aufnahmen ausgeschnitten mit Hilfe einer Maskenfunktion in DaVis (Kap.:3.2).

Die gespeicherten Aufnahmen einer Ziffer wurde dann mit dem `lvreader` (Kap.:3.1.3) in Python eingelesen und weiter bearbeitet.

Zuerst wurde alle Daten mit einer Zielvariablen (Label) versehen, dazu wurde die Aufnahmesets beschriftet mit der ersten Ziffer, die in dem Set zu sehen ist. Die Ziffern in einem Set laufen dann mit jeder Aufnahme eine Nummer (1,2,3...) weiter.

```
1 #creates array of labels
2 def addLabel ( file, label):
3     fileList1 = []
4
5     for f in os.listdir(file):
6         if f.endswith('.im7'):
```



```
7         fileList1.append(f)
8
9     count = len(fileList1)
10    start = int(file[-1])
11
12    for index in range(0, count):
13        digit = (start + index) % 10
14        label.append(digit)
```

Zur Reduzierung der Datengröße wurden die Aufnahmen auf ein Viertel der Originalgröße (zu  $112 \times 83$  Pixeln) minimiert und zur besseren Darstellung invertiert. Außerdem wurden sie in ein eindimensionales Array formatiert, so können sie direkt als Eingabewerte für ein Neuronales Netzwerk benutzt werden.

Zum Schluss wurden die Aufnahmen mit ihrem Label als numpy-Array gespeichert.

```
1 #convert to np array
2 labelNp = np.array(label, dtype= int)
3 dataNp = np.array(data, dtype= int)
4 dataSet = np.column_stack ((labelNp, dataNp))
5 print(dataSet.shape)
6 np.save("G:\data025", dataSet)
```

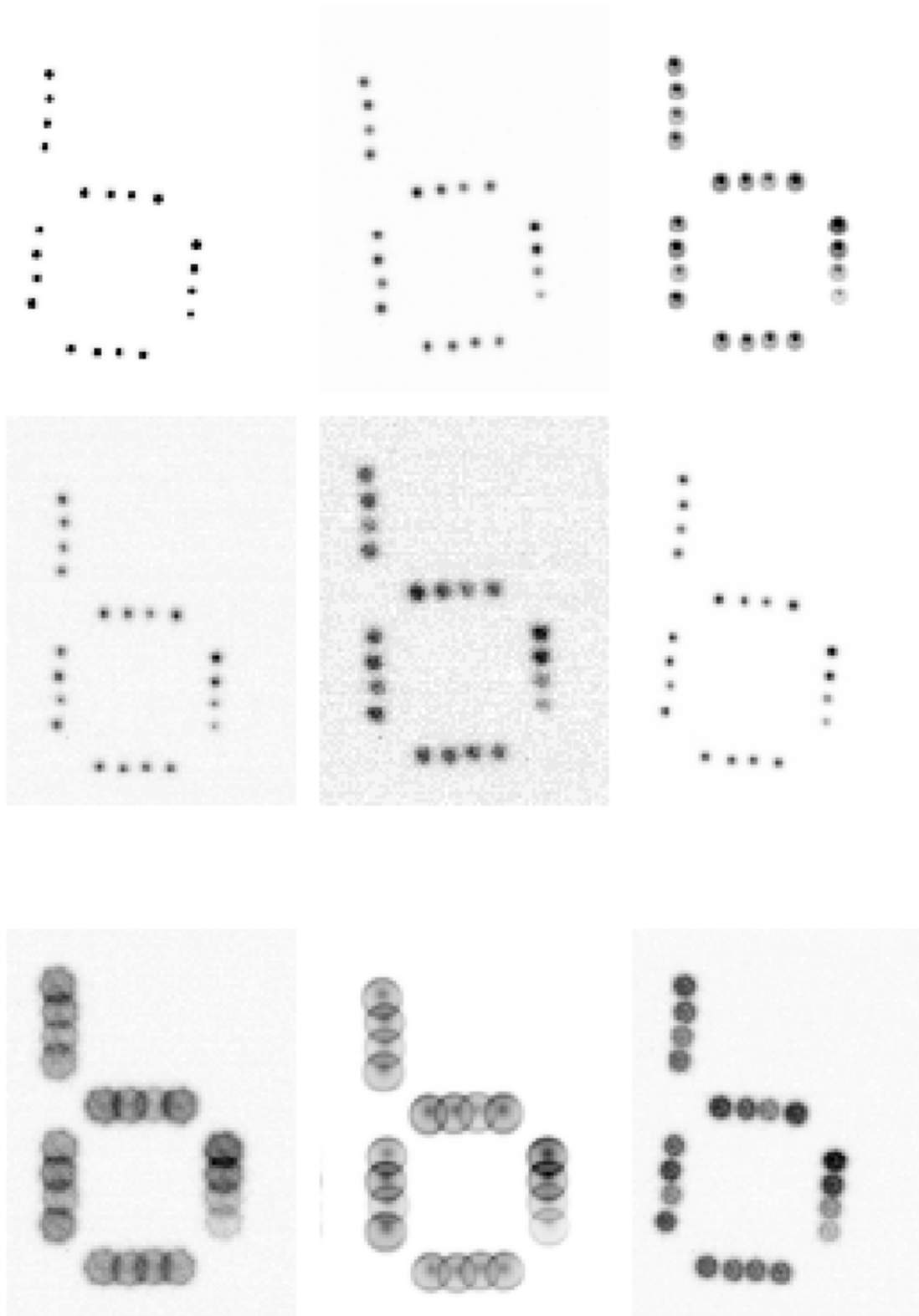


Abbildung 9: Beispiel der bearbeiteten Aufnahmen

## 4.3 Datenformat

Das Datenformat npy wurde gewählt, da es sich leicht und nativ in TensorFlow weiterverarbeiten lässt. Außerdem haben npy-Dateien eine schnellere Lesezeit und komprimieren die Daten besser als z.B. csv-Dateien. Beides ist hilfreich bei großen Datensätzen, wie sie für Maschinelles Lernen benötigt werden.[Url]

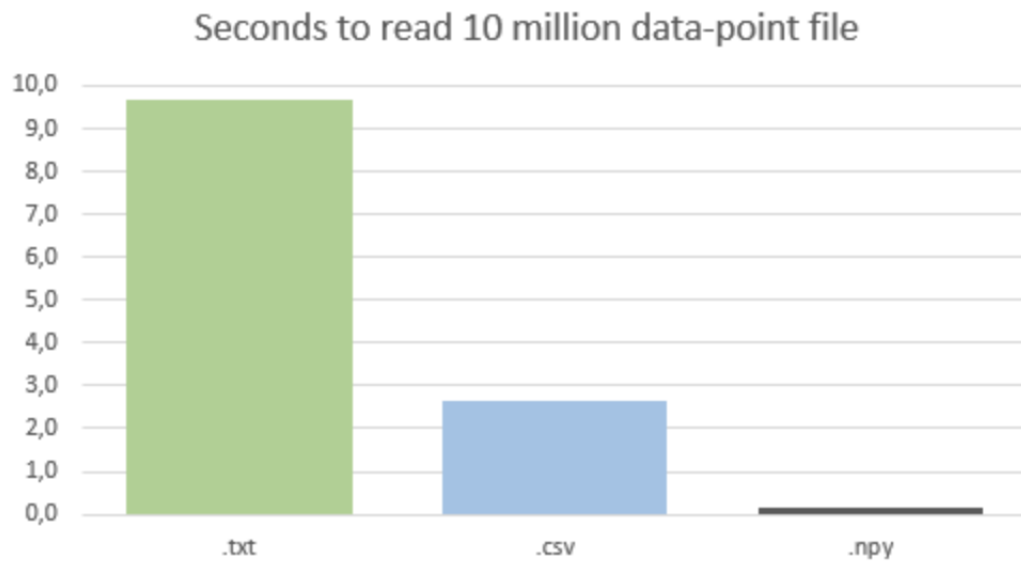


Abbildung 10: Lesegeschwindigkeiten der Datenformate [Url]

### 4.3.1 Datenteilung

Die Aufnahmen wurden in drei Datensätze unterteilt (Training-, Test- und Validierungssatz). Die Daten wurden vor der Unterteilung gemischt. Die Unterteilung der Daten geschieht, damit nicht mit dem gleichen Datensatz getestet wie gelernt wird. Damit lässt sich erkennen, ob der Algorithmus nicht-besondere nicht-relevante Eigenschaft erlernt hat.

Der dritte Datensatz dient zur Validierung. Nachdem ein Modell mit dem Trainieren abgeschlossen hat, kann nun an dem Validierungssatz getestet werden, wie gut das Modell mit ihm unbekannte Daten umgeht. Außerdem kann man unterschiedliche Modelle auf

ihre Effizienz mit einander vergleichen.

Damit ein Vergleich der unterschiedlichen Modelle möglich war, wurden die gemischten und dann getrennten Datensätze abgespeichert. Die neue abgespeicherten Datensätze wurden für das Lernen benutzt.

## 5 Modell

Auf der Suche nach einem guten Modell für die Daten wurden mehrere Ansätze gewählt und verfeinert. Als Grundlage wurde sich an Modellen für die Lösung des MNIST-Datensatz (circa 70.000 handgeschriebenen Ziffern) orientiert. Dieser Datensatz ist öffentlich verfügbar und enthält insgesamt 70.000 Graustufen-Bilder von handgeschriebenen Ziffern in  $28 \times 28$  Pixel. Er wird gerne als Beispiel für Einstieger verwendet.

### 5.1 Neuronales Netzwerk

Es wurde als erstes Neuronales Netzwerk ein sequentielles Modell benutzt. Dabei kann man mit Keras das Modell Schicht für Schicht aufbauen.

Das erste Modell besteht aus drei Schichten.

```
1 #keras model
2 model_seq = keras.models.Sequential()
3 model_seq.add(keras.layers.Dense(2000, activation='sigmoid',
    input_shape= [9296,]))
4 model_seq.add(keras.layers.Dense(100, activation='sigmoid'))
5 model_seq.add(keras.layers.Dense(10, activation="softmax"))
```

Mit einer 'keras.layers.Dense' Schicht wird jedes Neuron einer Schicht mit jedem Neuron in der Nachbarschicht verbunden. Als erstes wird die Anzahl der Neuronen in der Schicht festgelegt, danach ihre Aktivierungsfunktion und eine der möglichen weiteren Optionen beschreibt den Eingabevektor 'input\_shape'.

Der Eingabevektor von 9.296 entspricht einer Datengröße der Aufnahmen. Die erste Schicht nach der Eingabe hat 2.000 Neuronen. Es soll die Anzahl der Faktoren reduziert werden damit der Ausgabevektor zum Schluss des Modells mit der Anzahl der Zielvariablen übereinstimmt. Als Aktivierungsfunktion wurde die Sigmoid-Funktion gewählt. Die Sigmoid-Funktion gibt Werte zwischen Null und Eins zurück.

$$\text{sig}(w, b) = \frac{1}{1 + e^{\sum_j w_j z_j + b}} \text{ mit } j := \text{Neuron der Vorgängerschicht} \quad (10)$$

Sie hat den Vorteil, dass große Ausreißer in den Summe abgefangen werden mit ihrer Beschränktheit auf die Werte zwischen Eins und Null. Außerdem ist sie einfach und stetig differenzierbar. Die Ableitung der Sigmoid-Funktion lässt sich auf sich selbst zurück führen.

$$\text{sig}(w, b)' = \text{sig}(w, b)(1 - \text{sig}(w, b)) \text{ [Sig]} \quad (11)$$

Dieses erleichtert den Prozess des Lernens, da der Computer dabei die Differenzialgleichung auflösen muss.

Für die letzte Schicht wurde ein Softmax-Aktivierungsfunktion gewählt. Diese Funktion hat die Eigenschaft, dass die Summe der Ergebnisse in der Ausgabe-Schicht Eins ist. Also bekommt jedes Neuron als Ausgabe einen Wert zwischen Null und Eins. Gerade bei Klassifikationsaufgabe, wie der in dieser Arbeit, kann damit eine Wahrscheinlichkeit, das eine Klasse richtig ist, bestimmt werden.

Nachdem das Modell aufgebaut ist, wird im nächsten Schritt die Trainingsmethode festgelegt. Dazu werden unter anderem die Verlustfunktion, der Minimierungsprozess und die Lernrate bestimmt.

```

1 #model compiling
2 opt = tf.keras.optimizers.SGD(learning_rate=0.03)
3 model_seq.compile(optimizer=opt, loss=
    "sparse_categorical_crossentropy", metrics=['accuracy'])

```

Als Minimierungsprozess wurde die stochastischer Gradientenabstieg 'SGD' benutzt. Diese benutzt, wie in Kapitel 2.4.3 erklärt, den Gradienten für den Minimierungsprozess. Bei dem stochastischen Gradientenabstieg wird in einer Trainingseinheit immer ein Datensatz zufällig (stochastische) ausgewählt, um den Gradient nur über diesen einen Datensatz zu bilden. Das hat den enormen Vorteil, dass extrem viel Zeit und Rechenleistung gespart wird besonders im Vergleich zu dem normalen Gradientenabstieg. Hierbei wird der Gradient für jeden Datensatz in der Trainingseinheit gebildet. [Gé19]

Der Nachteil des stochastischen Gradientenabstiegs ist, dass es durch den zufälligen ausgewählten Datensatz nie zum global Minimum kommt. Bei einem hochdimensionalen Problem, wie die die durch Neuronale Netzwerke entstehen, gibt viele lokale Minima. Damit der Gradient sich besser an das global Minimum nähert, werden Mini-Batches benutzt. Dadurch wird das Trainingsset unterteilt und der stochastische Gradient in jedem dieser Stappel (Batch) gebildet. Dies führt zu weniger großen Sprüngen des Gradienten [Gé19].

Die Lernrate wurde mehrfach angepasst um das Modell zu optimieren. Die Verlustfunktion 'sparse\_categorical\_crossentropy' ist besonders nützlich für Klassifikationsaufgaben. Es wird ein Eingabevektor nur einer möglichen Ziffer zuordnet. Da es sich um eine Klassifikationsaufgabe handelt soll das Modell auf Genauigkeit evaluiert und trainiert werden, deswegen wurde 'metrics=['accuracy']' gewählt.

Danach wird das Modell trainiert, dazu müssen die Trainings- und Testdatensätze angegeben werden. Zusätzlich werden noch die Batch Größe und die Anzahl der Epochen festgelegt.

```

1 #Training and evaluating model
2 history = model_seq.fit(X_train,y_train, epochs= 10,
    validation_data=(X_test,y_test), batch_size=16)
3 print("fit history: ", history)
4
5
6 pd.DataFrame(history.history).plot(figsize=(8,5))
7 plt.grid(True)
8 plt.xlabel("Epochen")
9 plt.ylabel("Ergebnis")
10 plt.gca().set_ylim(0,1)
11 plt.show
12
13 model_seq.evaluate(X_val, y_val)

```

Der Befehl 'print(history)' zeigt den Lernprozess des Algorithmus an. Dieser kann außerdem in einer Graphik dargestellt werden.

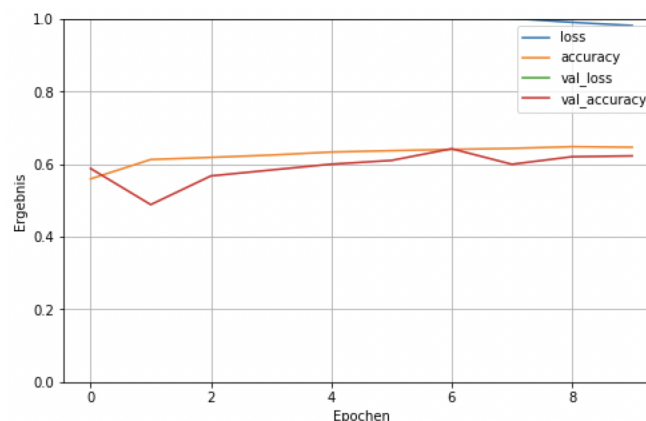


Abbildung 11: Lernprozess bei Lernrate 0.003 Batch= 16

### 5.1.1 Verbesserung

Im ersten Modell ist die Genauigkeit nur langsam gestiegen. Daraus wurde angenommen, dass die Lernrate zu klein ist. Deswegen wurde als nächstes die Lernrate auf 0.005 erhöht. Hier gab es stärker Schwankung bei der Genauigkeit der Testdaten. Um zu überprüfen,

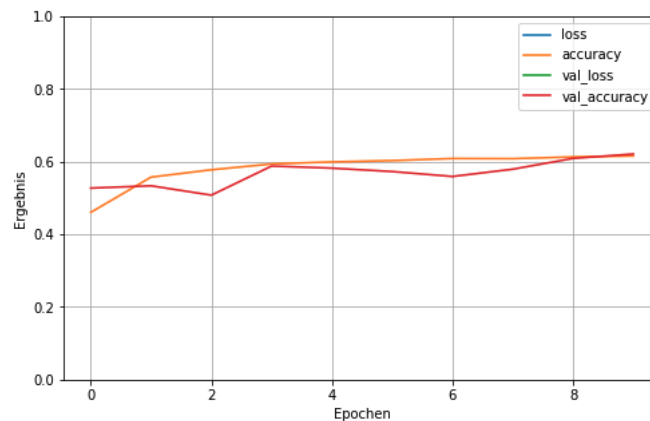


Abbildung 12: Lernprozess bei Lernrate 0.005 Batch= 16

ob die Lernphase einfach zu kurz war, wurde das ganze mit 30 anstelle von 10 Epochen wiederholt. Nachdem auch dort nur langsame Lernfortschritte gemacht wurden, wurde erneut versucht die Lernrate zu vergrößern. Sie wurde auf 0.01 gesetzt. Dies führt auch nicht zu einer deutlichen Verbesserung der Genauigkeit.

Als nächster Schritt wurde die Batch-Größe verkleinert auf 32. Das heißt der Trainingsdataset wird in 32 Teilstücke unterteilt und nach jedem Lernen mit einem Batch werden die Parameter angepasst. Das Unterteilen führt zu schnelleren Veränderung der Parameter

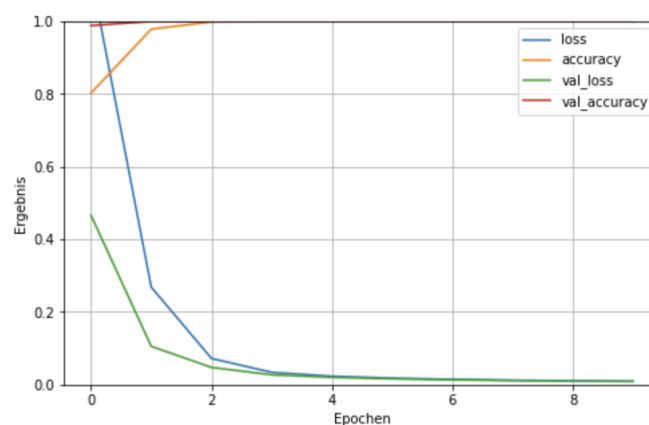


Abbildung 13: Lernprozess bei Lernrate 0.01 Batch= 32



hat aber auch den Nachteil, dass es zu Overfitting führen kann.

### 5.1.2 Überprüfung

Zur Kontrolle, ob der Algorithmus nicht unwichtige Besonderheiten gelernt hat, wurde einzelne Bilder nochmals per Hand überprüft.

```
1 test_model = tf.keras.models.load_model('drive/MyDrive/modelSeqSGD')
2 test_image = np.expand_dims(X_val[4], axis=0)
3 pre2 = test_model.predict(test_image)
4 pre2_max = np.argmax(pre2, axis = 1)
5 print('Image as Test. Label: {} Prediction {}'.format(y_val[4],
    pre2_max))
6 some_digit = X_val[4]
7 some_digit_image = some_digit.reshape(112,83)
8 plt.imshow(some_digit_image, cmap = mat.cm.binary,
    interpolation="nearest")
9 plt.axis("off")
10 plt.show()
```

Diese zeigten die Vorhersage und auch die Aufnahme wird dargestellt. Außerdem wurde

Image as Test. Label: 2 Prediction [2]

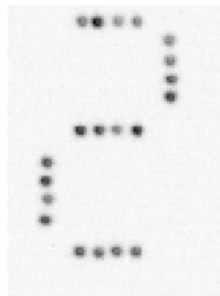


Abbildung 14: Überprüfung der Vorhersage

das Modell mit der Evaluierungsfunktion überprüft. Dazu wird das gefundene Modell mit seinen Parametern an dem Validierungsset getestet. Dieses Datenset kennt das Modell nicht, da die Daten nicht zum Lernen verwendet werden. Das Ergebnis war auch eine Genauigkeit von 100%.

## 6 Diskussion

In diesem Kapitel werden die Ergebnisse und Probleme des Maschinellen Lernens konkret auf diese Arbeit bezogen aber auch generell diskutiert. Außerdem wird ein Vergleich zwischen der Methode des Praxisprojektes (konventioneller Ansatz) und der Bachelorarbeit (ML-Ansatz) gezogen.

### 6.1 Ergebnis und Problem des ML

In der Arbeit ist aufgefallen, dass das Modell zu schnell zu gut funktioniert hat. Es wurde eine Erkennungsquote von 100 % erreicht. Es war vorher mit einer Quote um die 90 % gerechnet worden. Dieses wäre vergleichbar mit den ersten Ergebnissen des MNIST-Datensatzes gewesen.

Das zu gute Ergebnis hat zu einer langen Fehlersuche geführt. Es wurde überprüft, ob aus Versehen das Label mit in den Eingabewerten enthalten war und somit zum trainieren mitgegeben wurde. Dazu wurden stichprobenartig 150 Aufnahmen im Originalzustand nochmals per Hand angeschaut. Dabei wurde kontrolliert, ob nicht die Zielvariable in den Metadaten gespeichert wurde. (In der DaVis Software gibt es die Möglichkeit neben den Bilddaten auch Daten zum Bild, wie Kameratyp, Aufnahmezeit usw. zu speichern). Eine weite Überprüfung war die Ausgabe von Aufnahmen mit ihrem Label und der Vorhersage (s. Kpt. 5.1.2). Auch hierbei wurden keine Auffälligkeiten entdeckt. Es werden die richtigen Zahlen prognostiziert, die Zielvariable ist auch korrekt und die Ziffern sind nicht in einer Reihenfolge.

Als nächstes wurden zehn zufällige Aufnahmen ausgewählt um zu schauen, wie unterschiedlich die Aufnahmen tatsächlich sind. Das Problem könnte sein, dass der Unterschied zwischen den einzelnen Aufnahmen zu gering ist. Zum Beispiel sind alle Ziffern ungefähr gleich groß. Bei den Aufnahmen wurden keine unterschiedlichen Kameras oder Objektive benutzt.

Das Problem hierbei ist, dass es einfach ist für den ML-Algorithmus zu lernen und nicht genug generalisiert, um in andern Konfigurationen (Kameras und Objektiven) zu arbeiten.

Die Schwierigkeit, nicht ausreichend unterschiedliche Daten für einen ML-Algorithmus zu haben, ist auf der einen Seite durch die Unerfahrenheit beim Erstellen von Maschi-

nellen Lernen-Projekten entstanden. Hinzu kommt die voreingenommene Art Daten zu erheben. Es wurde zwar schon vorher über die benötigten Daten und die benötigte Unterschiedlichkeit der Aufnahmen nachgedacht und verschiedene Einstellungen verwendet, jedoch wurden auch wichtige Einstellungen vergessen.

Für eine Maschinelles Lernen-Projekt ist ein großer Satz von Daten, die schon vor dem Projekt erhoben wurden, nützlich, um den Einfluss der Voreingenommenheit zu verringern. Ein weiterer wichtiger Punkt, wenn die Daten schon vorhanden sind, ist der Zeitfaktor. Dieser kann deutlich reduziert werden, wenn die Daten schon vorhanden sind. Ein Nachteil von schon vorhanden Daten kann sein, dass sie nicht mit einem Label versehen sind. Das Versehen mit Zielvariablen ist eine aufwendige und zeitintensive Arbeit, wenn die Daten nicht in einer Reihenfolge oder schon mit irgendeinem Merkmal (Titel) markiert sind. In dieser Arbeit wurde die Reihenfolge der Aufnahmen genutzt, um möglichst einfach ein Label zu produzieren. Dieses geht natürlich nur, wenn die Reihenfolge bekannt ist. Eine Alternative wäre mit unüberwachtem Maschinellen Lernen zu arbeiten, um die Daten nicht mit einem Label versehen zu müssen.

## 6.2 Vergleich der Methoden

Die konventionelle Methode mit dem Durchlaufen der Segmente, wobei dann die Standorte und Helligkeit der LEDs im An-Zustand gespeichert werden, hat in dem Stand am Ende des Bachelor-Projektes eine Fehlerquote von 8,3 % [Bec21]. Diese Methode wurde noch weiter ausgearbeitet, um mit mehreren Kameras gleichzeitig zu arbeiten. Dabei wurde festgestellt, dass es in dem Algorithmus einen Logikfehler gab. Mittlerweile funktioniert auch diese Methode reibungslos. In dieser Bachelor-Arbeit wird der Zustand nach Ende des Bachelor-Praxis-Projektes verglichen, weil es sich beides um den ungefähren gleichen zeitlichen Aufwand hielt.

Der Vorteil der Methode ist, dass auf neue Kameras nicht extra eingegangen werden muss. Dadurch dass bei jeder Testreihe erst die Kalibrierung durchläuft, kann jedes Mal auf veränderte Umstände reagiert werden.

Zu den Nachteilen gehört, dass diese Art der automatischen Kalibrierung Schwierigkeiten mit stark rauschenden Aufnahmen hat. Der Grenzwert der Methode wird durch eine Formel festgelegt und es kann bei Aufnahmen, wo die Segmente im leuchtenden Zustand nur einige Counts höher liegen, Probleme bei der Erkennung geben.

Ein weiterer Nachteil ist, dass die Kalibrierung jedes Mal ausgeführt wird. Dieses kostet Zeit. Da die beanspruchte Zeit aber unter einer Minute für die gesamte Kalibrierung liegt und der Test jetzt komplett automatisch läuft, ist dieses kein Faktor mehr.

Die Maschinelle Lernen-Methode hat bei dem Daten zwar eine 100% Genauigkeit bei der Evaluierung, aber es wurde nur mit einer Kamera getestet. Die Erkennung funktioniert im Moment auch nur für eine Ziffer. Für eine echte Vergleichbarkeit müsste ein Algorithmus zur Bestimmung der Anzeige alle Ziffern betrachten werden. Die Daten für das Maschinelle Lernen und für die Fehlerquote für der konventionelle Methode wurden mit derselben Kamera und denselben Objektive aufgenommen und sind insofern vergleichbar.

Ein Nachteil des Maschinellen Lernens ist die Menge und Varianz der benötigten Lern-daten. So müssten für die unterschiedlichen Kameratypen und Objektive Aufnahmen gemacht werden, die zusätzlich noch jeweils in den genannten Parametern (4.1) variieren. Diese Arbeit würde wahrscheinlich weitere vier Wochen Zeit in Anspruch nehmen. Doch damit wäre immer noch nicht gesichert, ob das Ganze mit der nächsten neuen Kamera funktioniert. Dies ist aber kein großes Problem, da bevor eine neue Kamera bei LaVision benutzt wird, sie grundsätzlich sehr umfangreich getestet wird und in DaVis implementiert werden muss. Es lässt sich außerdem annehmen, dass das grundsätzliche Modell nicht neu aufgestellt werden muss, sondern nur erneut trainiert werden muss.

Ein weitere Nachteil ist die Verknüpfung der Aufnahme aus DaVis (C++) mit dem ML-Modell (Python). Außerdem müssten die Aufnahmen der einzelnen Ziffern alle in das richtige Maß für den Eingabevektor (in dem Model dieser Arbeit 9.296 Einträge) gebracht werden. Dieses ist zwar aufwendig, müsste aber nur einmal programmiert werden.

Der Vorteil des Maschinellen Lernens ist die Genauigkeit, die die Erkennung hat. So konnte in dieser Arbeit eine 100 % Quote nach der Evaluationsfunktion erreicht werden, ohne dass es irgendwelche Vorerfahrungen mit dem Maschinellen Lernen gab. Die Mustererkennung beim Maschinellen Lernen ist deutlich effizienter als konventionelle Ansätze, insbesondere auch, weil manche Muster extrem schwer zu beschreiben sind.

In dem behandeltem Beispiel für die automatische Kalibrierung und Erkennung einer Sieben-Segment-Anzeige war die konventionelle Methode einfacher und schneller. Die Fehlerquote ist aber höher und die gesamte Software und Infrastruktur ist für diesen Ansatz ausgelegt. So musste nur eine neue Kalibrierung entwickelt werden, den Algorithmus für die Erkennung gab es schon. Außerdem ist das Verarbeiten von der Aufnahmen bereits implementiert.

Es ist schwierig, bei so unterschiedlichen Eingangsvoraussetzung eine objektive Schlussfolgerung zu ziehen. Doch subjektiv gesehen, wenn es noch kein Lesealgorithmus für die Ziffern gegeben hätte, und eine Verknüpfung von DaVis zu einer Maschinen Lernen-Basis bestanden hätte, gehe ich davon aus, dass die Maschinellen Lernen-Methode effizienter gewesen wäre. Die Fehlerquote wäre geringer und der Mehraufwand des Datenaufnehmens ist eine Arbeit, die zwischendurch gemacht werden kann.

## 7 Ausblick

Während der Arbeit und Auswertung der Ergebnisse sind mehrere Ideen zur Fortsetzung und Verbesserung des Projektes entstanden.

Eine wichtige Voraussetzung, um den Teststand mit Maschinellern Lernen zu benutzen, wäre alle acht Ziffern zu erkennen. Dazu gäbe es zwei Methoden: entweder alle Ziffern mit einem ML-Algorithmus zu behandeln oder die Ziffern mit einer Methode zu trennen und einzeln in das, in dieser Arbeit, entwickelte Modell einzupflegen.

Alle Ziffern mit einem Maschinellen Lernen-Modell zu erfassen, wäre extrem aufwendig und datenintensiv. Unter der Annahme, dass mindestens 1.000 verschiedene Aufnahmen von der gleichen Darstellung gemacht werden müssten, und es bei acht Positionen mit je zehn verschiedene Ziffern  $10^8 = 100.000.000$  Darstellung gibt, sind das 100 Milliarden Aufnahmen. Bei einer Aufnahmegeschwindigkeit von  $22 \mu\text{s}$  sind das 25 Tage und dabei wurde noch nicht eingerechnet, dass Zeit gebraucht wird, um die Einstellungen zu ändern. Der reine Zeitfaktor macht diesen Ansatz nicht realistisch. Zusätzlich käme noch die Datenmenge hinzu. Der npy-File für dieses Projekt mit seinen 60.000 Aufnahmen im skalierten Zustand ist schon 3,5 GB groß. Wenn man von einem linearen Faktor bei der Datengröße aus geht, wären das 6.000 GB bei den skalierten Aufnahmen einer Ziffer.

Die zweite Variante wäre, aus den Aufnahmen die Ziffern zu trennen. Dazu gibt es unterschiedliche Methoden. In der Literatur wird oft ein MSER-Ansatz benutzt. Ein MSER ist ein Algorithmus mit dem Region und Kanten in Aufnahmen erkannt werden. Die andere Methode die Ziffern von einander zu trennen, würde auf dem bekannten Layout der Anzeige beruhen. Hierzu würden die Eck-LEDs erkannt werden und mit ihnen die Aufnahme in acht Rechtecke für die einzelnen Ziffern unterteilt werden. Diese Ausschnitte könnten dann mit dem Maschinellen Lernen-Modell erkannt werden.

Diese Vorgehensweise sollte ohne großen Aufwand zu implementieren sein. Die größte Schwierigkeit dabei wäre, dass DaVis auf C++ basiert und keine Bibliothek für Maschinellen Lernen besitzt. Die von DaVis aufgenommen Bilder müssten also in Python mit dem ML-Modell erkannt werden oder es müsste die TensorFlow Bibliothek in DaVis eingebaut werden.

## 7.1 DaVis

Beim Arbeiten an dem Maschinellen Lernen-Projekt sind mehrere Dinge aufgefallen, die in DaVis implementiert werden könnten, um die Möglichkeit zu schaffen Maschinelle Lernen-Projekt direkt in DaVis auszuführen. Als erstes wäre es sinnvoll, Aufnahmen mit Labels versehen zu können. DaVis besitzt viele Möglichkeiten, Aufnahmen zu bearbeiten und zu analysieren (z.B. Teilstücke ausschneiden und Histogramm der Helligkeitswerte). Es gibt in der Software schon die Option, Attribute zu den Aufnahmen hinzufügen und manche Information, wie Aufnahmezeit wird schon automatisch mit gespeichert. Ein Attribute zum Speichern des Labels wäre also ein geringer Aufwand. Vernünftig wäre die Zuordnungsmöglichkeit des Labels in alle aufgenommen Bilder einer Reihe, einzeln pro Bild oder in einer Schleifen-Funktion, so wie es für diese Arbeit benötigt wurde.

Weitere Überlegung müssten dazu angestellt werden, wie man das Maschinelle Lernen-Model mit DaVis verknüpft. Dazu gibt es drei Möglichkeiten:

Erstens die Aufnahmen werden nur in DaVis erstellt und bearbeitet und dann mit der `lvreader`-Bibliothek in Python eingelesen. In einem Python-Script also außerhalb von DaVis wird das Modell erstellt und mit dem Modell gearbeitet. Nachdem das Modell trainiert und gesichert wurde, müssen alle Aufnahmen, die von dem Modell bewertet werden sollen, in Python geladen werden. Dieses ist recht aufwendig für den Anwender ist aber mit dem aktuellen Stand von DaVis schon möglich.

Die zweite Möglichkeit wäre die Modelle immer noch in Python erstellen zu lassen, aber in DaVis eine Methode zu implementieren, um die Modelle zu laden und mit ihnen Aufnahmen zu analysieren. Dazu müsste die in Python geschriebenen Maschinellen Lernen-Modelle in der passenden Maschinellen Lernen-Bibliothek programmiert sein. Dazu bieten sich zum Beispiel die Bibliothek Pytorch [Pyt] oder TensorFlow an. Beide besitzen sowohl Bibliotheken in C++ als auch in Python und haben die Möglichkeit, Modelle aus Python in C++ zu laden [Pyt] [Ten].

Die dritte Möglichkeit wäre in DaVis ein Option zu implementieren, um ein Maschinelles Lernen-Modell direkt in DaVis zu bauen. Hierzu wäre es sinnvoll eine Maschinelle Lernen-Bibliothek in DaVis zu integrieren. Dann auf dieser Bibliothek beruhend eine Oberfläche zur Modell-Entwicklung in DaVis einzubetten. Geeignet wäre dazu sowohl TensorFlow als auch Pytorch (Torch als C++ Bibliothek) [Kol20].

## 8 Zusammenfassung und Fazit

Im Laufe dieser Arbeit wurde sich in Maschinelle Lernen eingearbeitet und ein Maschinelles Lernen-Modell für die Erkennung einer Ziffer einer Sieben-Segment-Anzeige erstellt. Diese Sieben-Segment-Anzeige mit acht weiteren Ziffern wird von der dem Highspeed-Kamera-Teststand der Firma benutzt, um die Qualitätssicherung von Kameras und Software durchzuführen. In der Arbeit wurde sich an die Vorgehensweise von Maschinellen Lernen-Projekt wie in s. 1.2 gehalten.

Es wurde sich erst mit der Problemstellung auseinander gesetzt. Die Anzeige sollte bei unterschiedlichen Einstellungen der Aufnahmeparameter s. 4.1 automatisch und verlässlich erkannt werden. Dieses ist gut gelungen auch, wenn nicht alle möglichen Einstellungen bedacht wurden Kapitel 6. Trotzdem kann an der Arbeit gut erkannt werden, wie zuverlässig sich ein Maschinelles Lernen-Modell für die Erkennung von einer Sieben-Segment-Anzeige trainieren lässt.

Damit das Modell überhaupt trainiert werden konnte, mussten als erstes Daten aufgenommen werden. Die Aufnahme der Daten geschah mit DaVis und die Daten wurden dann in Python weiterverarbeitet und mit einem Label versehen Kapitel 4. Dabei wurden die Aufnahmen skaliert und in ein numpy-File zusammengefasst, um die Datengröße zu verringern und schneller lesbar zu machen s. 10.

Zum Erstellen des Maschinellen Lernen-Projektes wurde sich für die Programmiersprache Python und ihre Bibliothek Keras sowie TensorFlow entschieden s. Kapitel: 3. Die Motivation, Python zu benutzen, kam daher, dass Python eine schon bekannte Programmiersprache ist und mit am häufigsten für Maschinelle Lernen-Projekte verwendet wird. Die Häufigkeit hat den Vorteil das viele Bücher und Beispiele in Python geschrieben sind. Gerade für jemanden, der sich in das Maschinelle Lernen einarbeitet, bietet Python und Keras einen schnellen, gut unterstützten und übersichtlichen Einstieg an. Außerdem lässt sich TensorFlow leicht in 'Google Colab' verwenden.

Danach wurde das Modell mit Keras erstellt. Bei dem Aufstellen des ML-Modelles wurde auf die Wahl der Aktivierungsfunktion geachtet. Zur Verbesserung des Modelles wurden die Lernrate und die Batch-Größe angepasst. Es stellte sich heraus, dass mit einer Vergrößerung der Batch-Größe, also der Unterteilung der Trainingsdaten in mehr Teilstücke, ein sehr gutes Ergebnis erzielt werden konnte. Die Funktion von Keras zum Evaluieren von dem Modell hatte eine Genauigkeit von 100% beim Erkennen des Validierungssets. Das sehr gute Ergebnis wird vor allem an der Ähnlichkeit der Aufnahmen gelegen haben



(s. Kapitel 6). Bei der Aufnahmen der Daten wurden wahrscheinlich nicht genug unterschiedliche Einstellung vorgenommen. Außerdem gibt es bei einer Ziffer einer Sieben-Segment-Anzeige deutlich weniger Unterschiede wie z.B. bei dem MNIST-Datensatz s.5. Dadurch ist Lernaufgabe für den Algorithmus unproblematisch. Damit das ML-Modell in der Praxis bei LaVision verwendet werden könnte, müsste noch mit Aufnahmen von unterschiedlichen Kameras und Objektiven trainiert werden.

Die Maschinelle Lernen-Methode wurde noch mit dem konventionellen Ansatzes der Bachelor-Praxis-Projektes verglichen, wobei dort der Standpunkt des Programmes zum Ende der Projektphase betrachtet wurde (s. Kapitel 6). Das Ergebnis des ML-Modelles war in diesem Zustand besser, aber es wurde auch nur eine Ziffer erkannt und der konventionelle Ansatz hatte auch noch einen Logikfehler. Nach einer Verbesserung funktioniert dieser nun sehr gut.

Zum Benutzen eines Maschinellen Lernen-Modelles mit der DaVis Software müsste noch eine Einbindung implementiert werden (Kapitel: 7). Dazu gäbe verschieden Möglichkeit: von der simplen Lösung die Aufnahmen in DaVis zu machen und in Python weiterzuverarbeiten, oder das Modell in Python zu erstellen und in DaVis ein Option zum Laden des ML-Modelles zu implementieren, oder in DaVis selbst eine Möglichkeit zum Erstellen von Maschinellen Lernen-Modellen zu programmieren.

Insgesamt war das Projekt ein guter Einstieg in das Maschinelle Lernen, so konnten alle Aspekte eines ML-Projektes s.1.2 durchgearbeitet werden und an ihnen Erfahrung gesammelt werden. Die Daten wurden selbstständig erzeugt und auch die Schwierigkeiten, die dadurch entstehen können wurden festgestellt. Außerdem war die Komplexität der Aufgabenstellung im richtigen Maße um erfolgreich ein erstes eigenes Maschinelles Lernen-Modell zu erstellen und erfolgreich zu verbessern. Der Umfang des Projektes hat auch gut gezeigt das Maschinelles Lernen für diese Art von Aufgabenstellung hervorragend geeignet ist.

Schlussendlich zeigt das Projekt, dass die Umsetzung von Maschinellen Lernen mit DaVis eine sinnvolle Überlegung für die Zukunft ist.

---

Ich würde gerne meinem Betreuer Jens Gutzeit von der Firma LaVision danken. Herr Gutzeit hatte immer Zeit und Geduld, mir bei meinen Fragen zu helfen.

Außerdem möchte ich mich noch bei mehreren Mitarbeitern von LaVision bedanken: bei Daniel Büchner für die Umprogrammierung der Anzeige, bei Dominik Schulz für die Änderung der Platine der Anzeige und bei Joachim Wolf für die Einführung in den Kamerateststand und die Erklärung aus Sicht eines Anwenders.

Des weiteren möchte ich mich noch bei Herrn Grothausmann für die Hilfe, einen Blick von außerhalb der Firma LaVision und die Grundidee für die Arbeit ein Maschinelles Lernen Projekt zu machen bedanken. Zusätzlich danke ich noch Mario Hesse für die Vorlage des  $\text{\LaTeX}$ -Dokumentes.

Außerdem möchte ich mich noch bei Kirsten Tuschling für das Korrektur lesen bedanken.

## Literaturverzeichnis

- [Bec21] Laura Luise Becker. “Automatische Kalibrierung eines Teststandes für Highspeed-Kameras”. Bachelor-Praxis-Projekt. Mai 2021.
- [Cho+20] Kenny Choo u. a. *Machine Learning kompakt*. Springer Fachmedien Wiesbaden, 2020. DOI: 10.1007/978-3-658-32268-7.
- [Gé19] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O’Reilly UK Ltd., Okt. 2019. 819 S. ISBN: 1492032646. URL: [https://www.ebook.de/de/product/33315532/aurelien\\_geron\\_hands\\_on\\_machine\\_learning\\_with\\_scikit\\_learn\\_keras\\_and\\_tensorflow.html](https://www.ebook.de/de/product/33315532/aurelien_geron_hands_on_machine_learning_with_scikit_learn_keras_and_tensorflow.html).
- [Kol20] Kirill Kolodiazhnyi. *Hands-On Machine Learning with C++*. Packt Publishing, Mai 2020. 530 S. ISBN: 1789955335. URL: [https://www.ebook.de/de/product/39149540/kirill\\_kolodiazhnyi\\_hands\\_on\\_machine\\_learning\\_with\\_c.html](https://www.ebook.de/de/product/39149540/kirill_kolodiazhnyi_hands_on_machine_learning_with_c.html).
- [Lav] *La Vision*. [www.lavision.de](http://www.lavision.de). URL: [www.lavision.de](http://www.lavision.de) (besucht am 01.09.2021).
- [Mat21] Stephan Matzka. *Künstliche Intelligenz in den Ingenieurwissenschaften*. Springer Fachmedien Wiesbaden, 2021. DOI: 10.1007/978-3-658-34641-6.
- [Pyt] URL: [https://pytorch.org/tutorials/advanced/cpp\\_export.html](https://pytorch.org/tutorials/advanced/cpp_export.html) (besucht am 21.09.2021).
- [Sig] URL: <https://mathworld.wolfram.com/SigmoidFunction.html-202109015> (besucht am 13.09.2021).
- [Ste18] Ralph Steyer. *Programmierung in Python*. Springer Fachmedien Wiesbaden, 2018. DOI: 10.1007/978-3-658-20705-2.
- [Ten] URL: [https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/label\\_image/main.cc](https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/label_image/main.cc) (besucht am 22.09.2021).
- [Urla] URL: <https://datasolut.com/was-ist-machine-learning/> (besucht am 01.09.2021).
- [Ur1b] URL: <http://neuralnetworksanddeeplearning.com/chap1.html> (besucht am 02.08.2021).
- [Ur1c] URL: <https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7> (besucht am 01.09.2021).

- 
- [Urld] URL: <https://numpy.org/doc/stable/user/what-is-numpy.html> (besucht am 07.09.2021).
- [Urle] URL: [https://scikit-learn.org/stable/getting\\_started.html](https://scikit-learn.org/stable/getting_started.html) (besucht am 07.09.2021).
- [UrLf] URL: <https://www.lavision.de/de/news/2021/4514/> (besucht am 01.09.2021).
- [UrLg] URL: <https://www.lavision.de/de/products/davis-software/index.php> (besucht am 07.09.2021).
- [UrLh] URL: <https://research.google.com/colaboratory/intl/de/faq.html> (besucht am 07.09.2021).
- [UrLi] URL: <https://de.wikipedia.org/wiki/C++> (besucht am 01.09.2021).
- [UrLj] URL: <https://towardsdatascience.com/what-is-npy-files-and-why-you-should-use-them-603373c78883> (besucht am 08.09.2021).

# Abbildungsverzeichnis

|    |   |    |
|----|---|----|
| 1  | Sechs . . . . .   | 7  |
| 2  | ML-Arbeitsablauf . . . . .                              | 8  |
| 3  | Neuron . . . . .  | 11 |
| 4  | Netzwerk Aufbau . . . . .                               | 12 |
| 5  | Neuronen in mehrschichtigen Netzwek . . . . .           | 12 |
| 6  | Aktivierungsfunktion . . . . .                          | 13 |
| 7  | Einfluss der Lernrate . . . . .                         | 15 |
| 8  | Highspeed-Kamera-Teststand . . . . .                    | 20 |
| 9  | Beispiel der bearbeiten Aufnahmen . . . . .             | 24 |
| 10 | Lesegeschwindigkeiten der Datenformate [Urlj] . . . . . | 25 |
| 11 | Lernprozess bei Lernrate 0.003 Batch= 16 . . . . .      | 29 |
| 12 | Lernprozess bei Lernrate 0.005 Batch= 16 . . . . .      | 30 |
| 13 | Lernprozess bei Lernrate 0.01 Batch= 32 . . . . .       | 30 |
| 14 | Überprüfung der Vorhersage . . . . .                    | 31 |