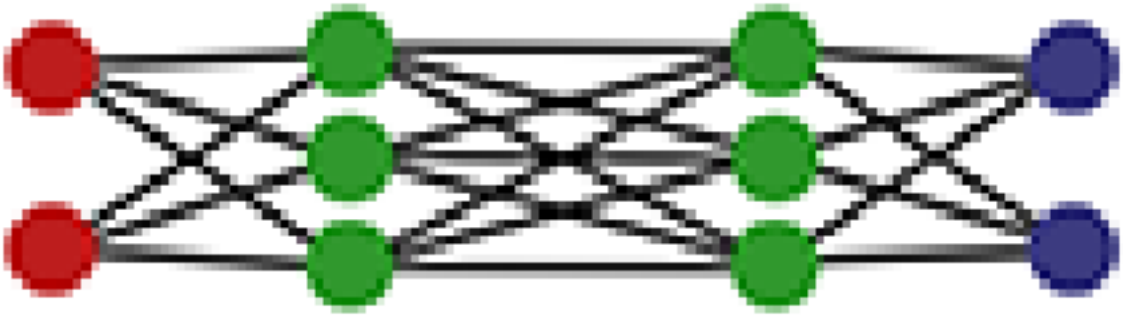
SCHRIFTENERKENNUNG MIT HILFE VON NEURAL NETWORKING



EINE ARBEIT VON
ARNE VIRIDEN

BETREUT VON
YVES WEBER

SCHRIFTENERKENNUNG MIT HILFE VON NEURAL NETWORKING

Eine Maturitätsarbeit an der

KANTONSSCHULE LIMMATTAL

vorgelegt von

ARNE VIRIDÉN

Klasse W6i

im Fach Informatik

betreut von

Yves Weber

2025

"Artificial Intelligence is no match for natural stupidity."

(Unknown, n.d.)

Inhaltsverzeichnis:

A. Einleitung

Programmieren ist seit meiner frühen Kindheit eine meiner liebsten Freizeitbeschäftigungen. Schon immer interessierten mich Algorithmen und lösungsorientiertes Denken, feste Bestandteile des Programmierens.

Seit den letzten Jahren besteht unumstritten eine deutlich stark gestiegene Aufmerksamkeit um Künstliche Intelligenz, angetrieben von grossen Modellen wie *ChatGPT* (Cretu, 2023) oder *Claude* (IBM, What is Claude AI?, 2024) welche beide auf Neural Networks basieren. Hierbei handelt es sich aber nicht um einen "gewöhnlichen Hype", sondern, meiner Meinung nach, um ein umfangreiches Forschungsgebiet mit vielversprechendem Potential. Aus diesem Grund fiel die Entscheidung, die eigene Maturitätsarbeit diesem Thema zu widmen.

B. Material

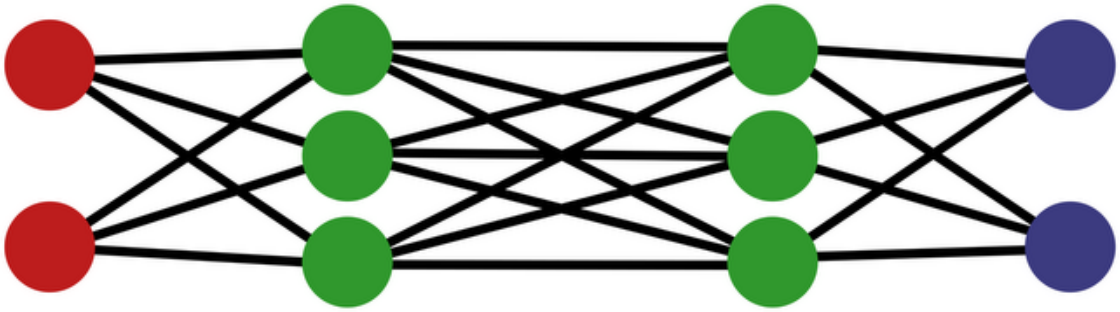
Für diese Arbeit wird die Python Version 3.9.6 verwendet, mit der externen Library *numpy*, in der Version 2.2.3.
Training des Neural Network, als auch das Ausführen des Programmes findet auf einem 2022 Apple Mac Studio mit M1 Max Chip statt.
Zudem wird das *EMNIST-Datenset* (NIST, 2024) für Trainings- und Testdaten für das Neural Network genutzt. (Cohen, Afshar, Tapson, & van Schaik, 2017)

Als Hauptquelle für das Lernen der Theorie und der Prinzipien hinter diesem Projekt wurden vor allem die YouTube Videos des Kanals "3Blue1Brown" (Sanderson, 3Blue1Brown, n.d.), insbesondere seine Serie über Neural Networks, welche einen simplen Überblick über deren Prinzipien vermittelt, gebraucht.

C. Neural Networks

C.1. Architektur und Forward Propagation

"... Ein neuronales Netz ist ein Programm oder Modell des maschinellen Lernens. Es trifft Entscheidungen auf ähnliche Weise wie das menschliche Gehirn: Seine Prozesse ahmen nach, wie biologische Neuronen zusammenwirken, um Phänomene zu identifizieren, Optionen abzuwägen und Schlussfolgerungen zu ziehen..." (IBM, Was sind neuronale Netzwerke?, 2021)



Visualisierung eines Neural Network Beispiels (eigene Darstellung)

Ein *Neural Network* besteht im Grossen und Ganzen aus verschiedenen *Layers*, welche wiederum aus Neuronen bestehen. Es sind jeweils alle Neuronen aus einem Layer, mit allen anderen aus einem anderen Layer, mit sogenannten *Weights* (in der obigen Abbildung in Schwarz dargestellt) verbunden.
Der erste Layer heisst *Input-Layer* (in Rot dargestellt), weil dieser den Input des NN (Neural Network) empfängt. Der letzte Layer nennt sich *Output-Layer* (in Blau dargestellt) und übergibt den Output des NN. Zu guter Letzt nennen sich alle Layers dazwischen *Hidden-Layers* (in Grün dargestellt), weil der Nutzer eines NN nicht wirklich Zugriff auf die Arbeitsweisen dieser Layer hat, sie sind so gesehen *hidden* (engl.; versteckt, verborgen) für den Anwender.

Ein NN lässt sich wie eine Küche vorstellen. Zutaten werden in die Küche gebracht (Input-Layer). Köche (Neuronen), an verschiedenen Stationen verarbeiten die Zutaten auf ihre ganz eigene Weise (*Biases*). Jeder Arbeitsschritt des Kochens ist Teil einer bestimmten Reihenfolge (Layers). Jedes Gewürz ist unterschiedlich stark (*Aktivierungsfunktionen*) und hat einen verschieden starken Einfluss (Weights) auf das finale Mahl (Output-Layer). Schmeckt das fertige Gericht nicht, sorgen Anpassungen in ganz bestimmten Schritten der Zubereitung zukünftig für ein optimaleres Resultat (Backward Propagation, wird in späteren Kapiteln weiter angesprochen).

Doch was sind Weights, Biases und Aktivierungsfunktionen genau, und wofür werden diese überhaupt in der Forward Propagation gebraucht?

Weights machen genau das, was ihre Übersetzung ins Deutsche vermuten lässt. Die *Gewichte* (dt.; Weights) gewichten, wie wichtig ein Neuron und dessen Wert ist. Sie beeinflussen die Abhängigkeit zwischen zwei Neuronen, da sie als eine Art Brücke agieren. Werte von Neuronen, welche auf andere übertragen werden, werden bei der Forward Propagation ganz einfach mit den dazugehörigen Weights multipliziert. Das bestimmt den Grad der Auswirkung, welchen ein Neuron auf einen anderen hat.

Wie bei den Weights, bleiben auch die Biases ihrem Namen treu. Als Konstante, welche den Neuron-Werten unabhängig vom Input addiert wird, verleihen sie dem betroffenen Neuron eine

Art Bias. Besonders wichtig können Biases im Zusammenspiel mit Aktivierungsfunktionen sein.

Aktivierungsfunktionen sind *nicht lineare Funktionen*, durch welche Neuron-Biases addiert mit Neuron-Inputs in Abhängigkeit von den Weights zu den Neuron-Werten werden. Sie bestimmen also den finalen Neuron-Wert. Aktivierungsfunktionen können sehr viel Komplexität zu den Neural Networks hinzufügen, was diesen erlaubt, noch komplexere Muster zu erkennen.

Zu den bekanntesten Aktivierungsfunktionen zählen:

1. *ReLU*

$$f(x) = max(x, 0)$$

C.1.a

2. *Leaky ReLU*

$$f(x, a) = max(x, x \times a);\ 0 < a < 1$$

C.1.b

3. *Sigmoid*

$$f(x) = \frac{1}{(1 + e^{-x})}$$

C.1.c

4. *Softmax*

$$f(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

C.1.d

Im Allgemeinen sind NN komplex, um möglichst anpassungsfähig zu sein und um eine grössere Lernkapazität zu besitzen. Aus dem Grund benötigen diese einige unterschiedliche Elemente, wie in diesem Fall Weights, Biases, und Aktivierungsfunktionen.

Die Berechnung des Outputs eines NN erfolgt durch die sogenannte *Forward Propagation*. Hierbei gilt folgendes für Neuronen in Hidden- und Output-Layers, wobei l für den Layer, w für das Weight, b für den Bias, n für das Neuron und ϕ für die Aktivierungsfunktion stehen:

$$a_{l,n} = \phi(b_{l,n} + \sum_{k}^{\#n\ im\ l-1} a_{l-1,k} \times w_{l,k,n})$$

C.1.e

Im Input-Layer gilt:

$$a_{l,n} = \phi(b_{l,n} + Input_{l,n})$$

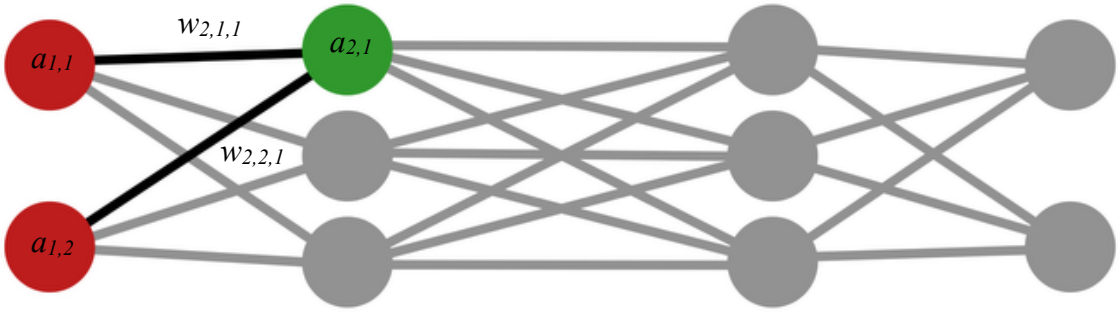
C.1.f

$a_{l,n}$ ist hier der berechnete Wert eines Neurons, wobei l für den Layer steht, in welchem sich das Neuron befindet und n für die Stelle, respektive Zeile, an welcher sich das Neuron in seinem Layer befindet. Dasselbe gilt für $b_{l,n}$bezüglich Biases und $z_{l,n}$, wobei:

$$a_{l,n} = \phi(z_{l,n})$$

C.1.g

Bei $w_{l,k,n}$ steht l für den Layer, zu dem das Weight gehört, k für die Position des einen Neuronen im vorherigen Layer und n für die Position des anderen verbundenen Neuronen im dazugehörigen Layer.



Visualisierung der Forward Propagation (eigene Darstellung)

Für das Beispiel in der obigen Abbildung wäre die Berechnung für das erste Neuron des zweiten Layers daher:

$$a_{1,1} = \phi(Input_{1,1} + b_{1,1})$$

$$a_{1,2} = \phi(Input_{1,2} + b_{1,2})$$

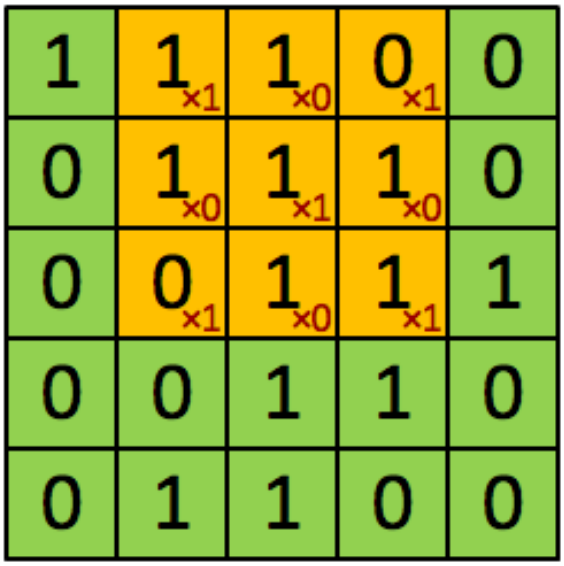
$$a_{2,1} = \phi(a_{1,1} \times w_{2,1,1} + a_{1,2} \times w_{2,2,1} + b_{2,1})$$

C.2. Neural Network Typen

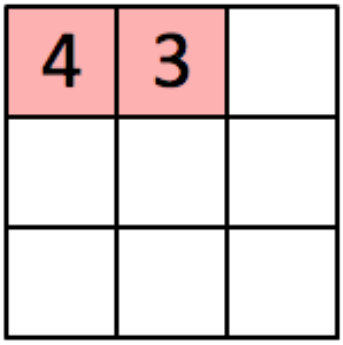
Das Beispiel, welches im vorherigen Kapitel behandelt wurde und weiterhin im Fokus stehen wird, ist ein sogenanntes *FCNN* (Fully Connected Neural Network). Ein solches besteht aus dem vorhin erläuterten 3 Teile System, aus Input-, Hidden- und Output-Layers, in welchen die verschiedenen aufeinanderfolgenden Layers jeweils vollständig miteinander verbunden sind, also Fully Connected. Das FCNN ist wahrscheinlich das Bekannteste/Geläufigste von allen Typen, da es auch das Simpelste und Grundlegendste ist.

Es gibt jedoch noch einige weitere Variationen von NN, wie zum Beispiel *CNN* (*Convolutional Neural Network*) oder *RNN* (*Recurrent Neural Network*) und viele mehr.

CNN ist heutzutage einer der wichtigsten NN-Typen in einer digitalen Welt voller Medien wie Bilder und Töne. Denn genau in diesem Gebiet glänzt es: Die Stärke der CNN ist es, wichtige Merkmale aus einem Pool von Rohdaten herauszupicken.
Aufgebaut sind CNN ähnlich wie FCNN, nur mit dem Unterschied, dass ganz am Anfang des CNN mindestens ein *Convolutional Layer*, und manchmal einige *Pooling Layers* liegen. Der Input wird direkt in diese Layers eingegeben, und die Outputs dieser Layers kommt anschliessend in ein FCNN.
Ein Convolutional Layer besteht aus einem sogenannten *Filter* (auch *Kernel* genannt). Ein Filter ist eine Matrix, bestehend aus Variablen, welche bei der Forward Propagation über den Input "geschoben" wird. Das Skalarprodukt aus diesem Filter und der "darunterliegenden" Matrix aus dem Input ergibt anschliessend einen Output Wert.



Input



Output

(University, 2025)[eigene Bearbeitung]

Bei Pooling Layers wird jeweils von einer Region entweder der Durchschnitt der darunterliegenden Werte entnommen, ein Maximalwert, ein Minimalwert oder eine andere Eigenschaft.

Das Ziel des CNN ist es, einen Kontext zwischen zusammenhängenden Werten spezifisch zu nutzen und die räumlichen Dimensionen des Inputs gezielt auf das Wichtigste zu reduzieren. Pooling Layers erlauben mithilfe von Maxima und Minima einen weiteren Aspekt der Komplexität, mit welchem ein CNN noch mehr Lernpotential besitzt und markante Eigenschaften herauslesen kann.

RNN gehören ebenfalls zu den wichtigsten NN-Typen. Im Gegensatz zu traditionellen NN beziehen RNN vorherige Inputs und Outputs einer Sequenz bei Berechnungen mit ein. Sie besitzen eine Art Gedächtnis und werden daher oftmals für Übersetzungen oder in der Meteorologie (Jung Min Han, 2021) eingesetzt. Die Wichtigkeit des Gedächtnisses ist ebenfalls dank Backward Propagation flexibel und erlaubt dem RNN beispielsweise relative Zeit im Kontext mit anderen Events als Variable bei Berechnungen zu nutzen. Auch dieses Element verschafft einem RNN eine überlegene Kapazität zur Komplexität im Vergleich zu herkömmlichen NN.

Ausserdem lassen sich verschiedene NN-Typen auch unterschiedlich kombinieren. Das heisst, es wäre beispielsweise möglich einen Convolutional Layer zu einem RNN hinzuzufügen und NN je nach Anwendungsbereich zu spezifizieren.

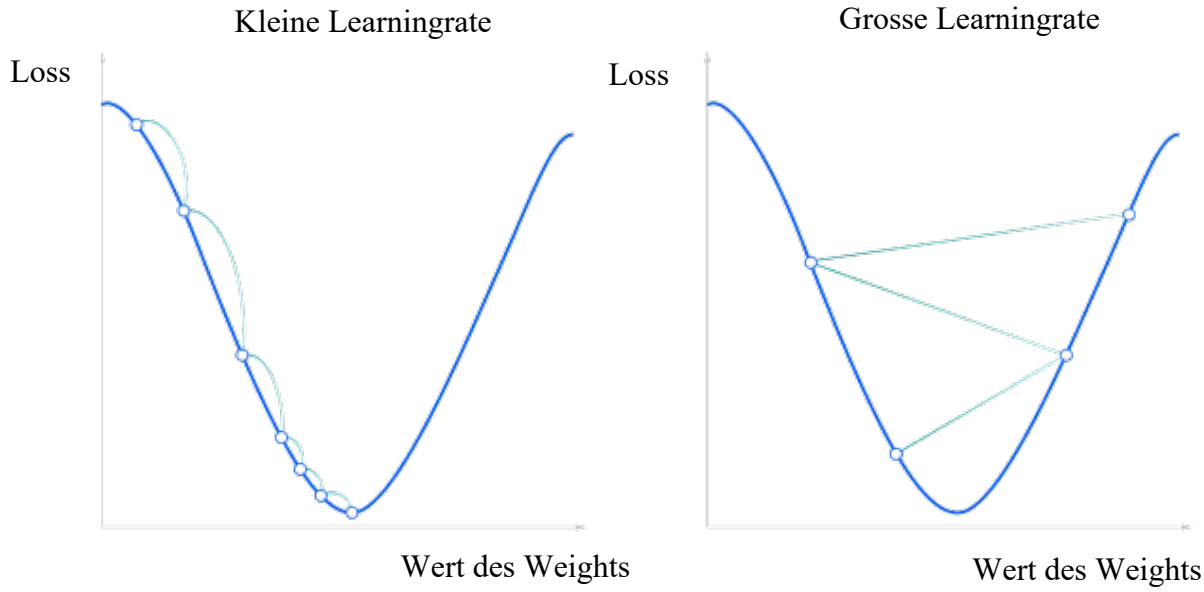
Da jedoch diese beiden genannten anderen Typen das Programm in meinem Fall leider nur komplizierter und unpraktisch (ein RNN wäre zu ineffizient, ein CNN wäre wegen der Input-Grösse nicht passend) machen und andere NN-Arten, als auch Kombinationen, das Erklären nur verkomplizieren würden, wurde darauf verzichtet, diese zu implementieren.

C.3. Backward Propagation

Die Backward Propagation ist ein fundamentaler Algorithmus, den NN zum "Lernen" benötigen. Hierbei werden Fehler, die das NN gemacht hat, mithilfe von *Gradienten* zurückverfolgt.
Das heisst, ist ein Output fehlerhaft, wird versucht mithilfe von Backward Propagation herauszufinden, welche Weights und Biases (in einem FCNN, in anderen NN-Typen würden noch mehr Elemente hinzukommen, die bei der Backward Propagation miteinbezogen werden würden) vorwiegend an diesem fehlerhaften Output schuld sind. Diese Weights und Biases werden anschliessend um ein Stück verändert.

Der Gradient ist eine Art Steigung für jedes Element in einem NN, er ist die Veränderung des Outputs im Vergleich zur Veränderung am Wert des Elements selbst. Dieser zeigt in dem Sinne den Einfluss eines Elements auf den Output an. In der Backward Propagation wird dieses Phänomen genutzt, um bei falschen Outputs die verantwortlichen Elemente anzupassen, sodass der Output näher am gewünschten Output liegt. Je grösser der Gradient eines Elements, desto schuldtragender ist dieses bezüglich des fehlerhaften Outputs.
Wiederholt sich dieser Prozess, passt sich das NN langsam an Muster in den Datensätzen an.

Wie schnell dieser Prozess abläuft, hängt von der sogenannten *Learningrate* (η) ab. Dies ist ein Faktor, welcher die Veränderung an Weights und Biases beeinflusst. Eine kleine Learningrate sorgt für kleinere Veränderungen im NN während der Backward Propagation, eine grössere Learningrate für grössere.



(Adari, 2023)[eigene Bearbeitung]

Die obige Abbildung visualisiert den Backward Propagation Prozess. Jeder Punkt auf der Linie symbolisiert eine durchgeführte Backward Propagation an einem NN, in der Reihenfolge der verbundenen Linien. Hier wird der Ablauf des Annäherns an ein lokales Minimum veranschaulicht, das Ziel der Backward Propagation, über mehrere Wiederholungen den Gesamtfehler (*Loss*) des NN schrittweise zu minimieren.

Der Loss ist eine Methode, um auszurechnen, wie falsch das NN liegt. Eine der allgemein bekanntesten Loss-Funktionen ist der *Quadratic Loss*:

$$\frac{\overbrace{\partial C}^{Symbol\ für\ den\ Fehler}}{\partial a_{l,n}} = 2(a_{l,n} - y)$$

C.3.a

Die gebräuchlichste Loss-Funktion für Klassifikationsprobleme ist jedoch die *Cross-Entropy Loss Function* (gilt für Neuronen im Output-Layer mit Softmax), welche verwendet wird:

$$\frac{\partial C}{\partial z_{l,n}} = a_{l,n} - y_n$$

C.3.b

Diese wird in der Regel immer in Kombination mit der Softmax-Aktivierungsfunktion (Formel C.1.d) genutzt und passt daher perfekt zum NN.

Dazu kommt, dass Softmax im Vergleich zu einem Arrangement von Neuronen mit Sigmoid-Aktivierungsfunktion eine Wahrscheinlichkeitsverteilung ausgibt, wobei letzteres Klassen als unabhängig behandelt.

Für die Backward Propagation gilt (Mazur, 2015) (Sanderson, Backpropagation calculus | Deep Learning Chapter 4, 2017) durch Aufspalten der einzelnen Elemente mithilfe der Kettenregel:

Um bei der Backward Propagation den Fehler der Weights zu berechnen, wird mithilfe von Gleichung C.1.e die angehende Gleichung abgeleitet:

$$\frac{\partial a_{l,n}}{\partial z_{l,n}} = \phi'(b_{l,n} + \sum_{k}^{\#n\ im\ l-1} a_{l-1,k} \times w_{l,k,n})$$

C.3.c

Ebenfalls wird aus der Gleichung C.1.e differenziert:

$$\frac{\partial z_{l,n}}{\partial w_{l,i,n}} = a_{l-1,i}$$

C.3.d

Direkt besteht keinen Zugriff auf den Fehler der Weights, jedoch ist es möglich diesen in Faktoren aufzuteilen, welche bekannt sind. Dadurch gilt im Allgemeinen für Weights:

$$\frac{\partial C}{\partial w_{l,i,n}} = \frac{\partial z_{l,n}}{\partial w_{l,i,n}} \times \frac{\partial a_{l,n}}{\partial z_{l,n}} \times \frac{\partial C}{\partial a_{l,n}}$$

C.3.e

Die Gleichungen C.3.c, C.3.d und C.3.m lassen sich nun in der Gleichung C.3.e einsetzen, wodurch eine Schlussformel für den Fehler der Weights resultiert:

$$\Rightarrow \frac{\partial C_{l,n}}{\partial w_{l,i,n}} = a_{l-1,i} \times \phi'(b_{l,n} + \sum_{k}^{\#n\ im\ l-1} a_{l-1,k} \times w_{l,k,i}) \times \frac{\partial C}{\partial a_{l,n}}$$

C.3.f

Um bei der Backward Propagation den Fehler der Biases zu berechnen, wird mithilfe von Gleichung C.1.e die nachstehende Gleichung abgeleitet:

$$\frac{\partial z_{l,n}}{\partial b_{l,n}} = 1$$

C.3.g

Allgemein gilt für Biases, nach dem gleichen Prinzip wie betreffend der Weights:

$$\frac{\partial C}{\partial b_{l,n}} = \frac{\partial z_{l,n}}{\partial b_{l,n}} \times \frac{\partial a_{l,n}}{\partial z_{l,n}} \times \frac{\partial C}{\partial a_{l,n}}$$

C.3.h

Die Gleichungen C.3.c, C.3.g und C.3.m werden in der Gleichung C.3.h eingesetzt, auf welcher Weise die Schlussformel für den Fehler der Biases entsteht:

$$\Rightarrow \frac{\partial C}{\partial b_{l,n}} = \phi'(b_{l,n} + \sum_{k}^{\#n\ im\ l-1} a_{l-1,k} \times w_{l,k,n}) \times \frac{\partial C}{\partial a_{l,n}}$$

C.3.i

Um bei der Backward Propagation den Fehler der Neuronen zu berechnen, wird mithilfe von Gleichung C.1.e die folgende Gleichung abgeleitet:

$$\frac{\partial z_{l+1,i}}{\partial a_{l,n}} = w_{l+1,n,i}$$

C.3.j

Auch der Fehler der Neuronen lässt sich in bekannte Faktoren aufteilen, es gilt in der Theorie für Neuronen in einem NN bestehend aus einem Neuron pro Layer folgende Gleichung:

$$\frac{\partial C}{\partial a_l} = \frac{\partial z_{l+1}}{\partial a_l} \times \frac{\partial a_{l+1}}{\partial z_{l+1}} \times \frac{\partial C}{\partial a_{l+1}}$$

C.3.k

Diese Gleichung C.3.k wird nun für spezifische Neuronen in einem NN, in welchem Layer aus mindestens einem Neuron bestehen, umformuliert. Ein Neuron erbt nun nicht nur einen Teil des Fehlers eines Neurons, sondern Teile der Fehler aller vorhergehenden Neuronen, dargestellt als Summe:

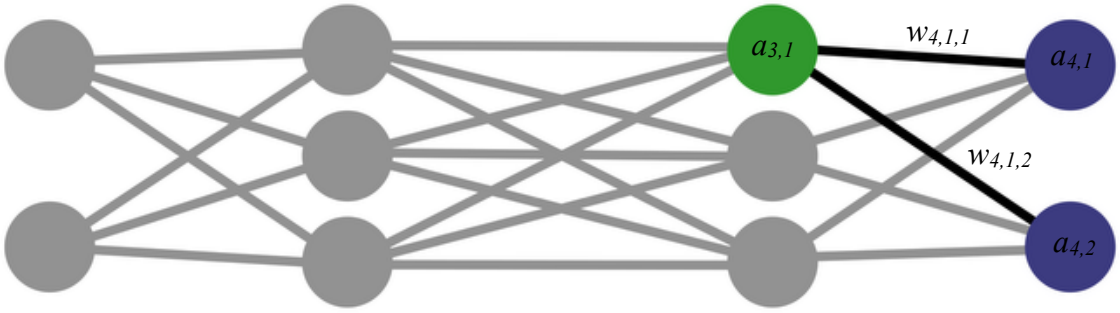
$$\Rightarrow \frac{\partial C}{\partial a_{l,n}} = \sum_{i}^{\#n\ im\ l+1} \frac{\partial z_{l+1,i}}{\partial a_{l,n}} \times \frac{\partial a_{l+1,i}}{\partial z_{l+1,i}} \times \frac{\partial C}{\partial a_{l+1,i}}$$

C.3.l

Die Gleichungen C.3.c, C.3.j werden in der Gleichung C.3.l eingesetzt:

$$\Rightarrow \frac{\partial C}{\partial a_{l,n}} = \sum_{i}^{\#n\ im\ l+1} w_{l+1,n,i} \times \phi'(b_{l+1,i} + \sum_{k}^{\#n\ im\ \ l} a_{l,k} \times w_{l+1,k,i}) \times \frac{\partial C}{\partial a_{l+1,i}}$$

C.3.m



Visualisierung der Backward Propagation (eigene Darstellung)

Fürs Trainieren von NN gibt es viele Möglichkeiten. Die bekannteste ist wahrscheinlich der *Mini-Batch Gradient Descent*.
Hierbei teilt man all seine Trainingsdaten in kleinere Sets, sammelt alle Fehler der einzelnen Datendurchläufe innerhalb dieses Sets zusammen, und korrigiert das NN anschliessend, also einmal pro Set insgesamt. Dadurch schwankt der Lernprozess des NN weniger als beispielsweise beim *Stochastic Gradient Descent*, bei welchem man nach jedem einzelnen Datenwert Backward Propagation betreiben würde. Oftmals trainiert man sein NN wiederholt mit den gleichen Trainingsdaten, in mehreren Durchläufen. Ein solcher Durchlauf mit den gleichen Trainingsdaten nennt sich *Epoch*.

C.4. Weiteres

NN sind ein reicher, vielschichtiger Gegenstand. Dadurch gibt es einige Probleme, welche man beim Training von NN antreffen kann und welche es zu verhindern oder zu begrenzen gilt.
Zwei der wichtigsten: *Overfitting* und *Underfitting*. Vor allem der Erstere stellt ein grösseres Problem dar.

Das Hauptziel von NN an sich ist das Erkennen von Mustern aus Datensätzen, um dann akkurate Vorhersagen zu unbekannten Werten zu tätigen. Das Lernen dieser Muster ist bei NN, wie beim Menschen eine Art Prozess oder Spektrum, da es unendliche von Möglichkeiten gibt, verschiedene Details und Verallgemeinerungen unterschiedlich zu gewichten. Bei NN gibt es oftmals bezüglich gewissen Hyperparametern oder Elementen Schwachstellen oder Fehlerquellen, welche das erwünschte Erreichen einer optimalen Balance zwischen Detail und Verallgemeinerung, also der erzielten Mustererkennung, erschweren oder sabotieren. Das Ergebnis: Zwei Extreme des Spektrums. Overfitting und Underfitting. Fixation auf Details und blinde Generalisierung.

Overfitting geschieht oft aufgrund von einem zu komplexen NN im Vergleich zur Datenkomplexität und Datenmenge. Beim NN-Training werden oftmals Datensätze wiederholt zum Training eingesetzt, was dem NN die Möglichkeit bietet, die "Antworten", also Outputs, auf gewisse Inputs einfach "auswendig zu lernen", anstatt ein Muster zu erkennen.

Underfitting entsteht, wenn das NN im Vergleich zu den Datensätzen schlichtweg "zu simpel" ist, also zu wenig komplex. Das macht das Lernen unmöglich und das NN gibt anschliessend nur noch ungenaue oder bedeutend fehlerhafte Outputs, da das Muster nicht erkannt werden konnte.

Auf der anderen Seite gibt es aber einige Elemente oder Strategien, die sich in NN einbauen lassen, welche gewissen potenziellen Problemen vorbeugen oder diese reduzieren.

Eines der Elemente nennt sich *Dropout*. Eine Methode, bei welcher im Training stückweise zufällige Neuronen in einem NN ausfallen gelassen werden und so Forward- und Backward Propagation durchgeführt wird. Das Resultat: Neuronen sind "unabhängiger" von anderen Neuronen, da diese nicht mehr immer zuverlässig da sind. Daraus folgt Verallgemeinerung des Inputs, da die "Gedankengänge" des NN nicht mehr komplex genug für das Auswendiglernen sind. Ausserdem reduziert Dropout auch die Anzahl NN pro Durchlauf, das NN wird kleiner, also weniger komplex, was auch Overfitting reduziert oder gar vorbeugt. Man kann den Dropout mithilfe von Parametern beschreiben, welche den Anteil der ausfallenden Neuronen in einem bestimmten Layer symbolisieren.

Eine weitere wichtige Praxis wäre die *Regularisierung*. Regularisierung versucht, wie Dropout, Overfitting zu reduzieren oder gar vorzubeugen, indem es zu grosse Absolutwerte bei den Weights an 0 anzunähern versucht. Das funktioniert, da Overfitting ein NN benötigt, welches sehr sensibel gegenüber kleinen Werteveränderungen im Input ist, welche es so auswendig lernen kann. Diese Sensibilität lässt sich auf die Weights zurückweisen, da deren Auswirkungen am stärksten vom Input abhängig sind. Es gibt verschiedene Arten von Regularisierung, welche verschiedene Zwecke erfüllen (werden nicht weiter elaboriert).

Die Art, die für das NN genutzt wird, ist die *L2-Regularisierung*, auch genannt *Ridge Regression* (Ian Goodfellow, 2016):

$$\partial C \rightarrow \partial C + \frac{\lambda}{2 \times \underbrace{n}_{Mini-Batch-Grösse}} \times \sum w^2$$

C.4.a

Löst man die obige Formel nach dem Fehler des Weights im Verhältnis zum totalen Fehler auf, erhält man folgende Gleichung für die Backward Propagation:

$$\frac{\partial C_{l,n}}{\partial w_{l,i,n}} \rightarrow \frac{\partial C_{l,n}}{\partial w_{l,i,n}} + \frac{\lambda}{n} \times w_{l,i,n}$$

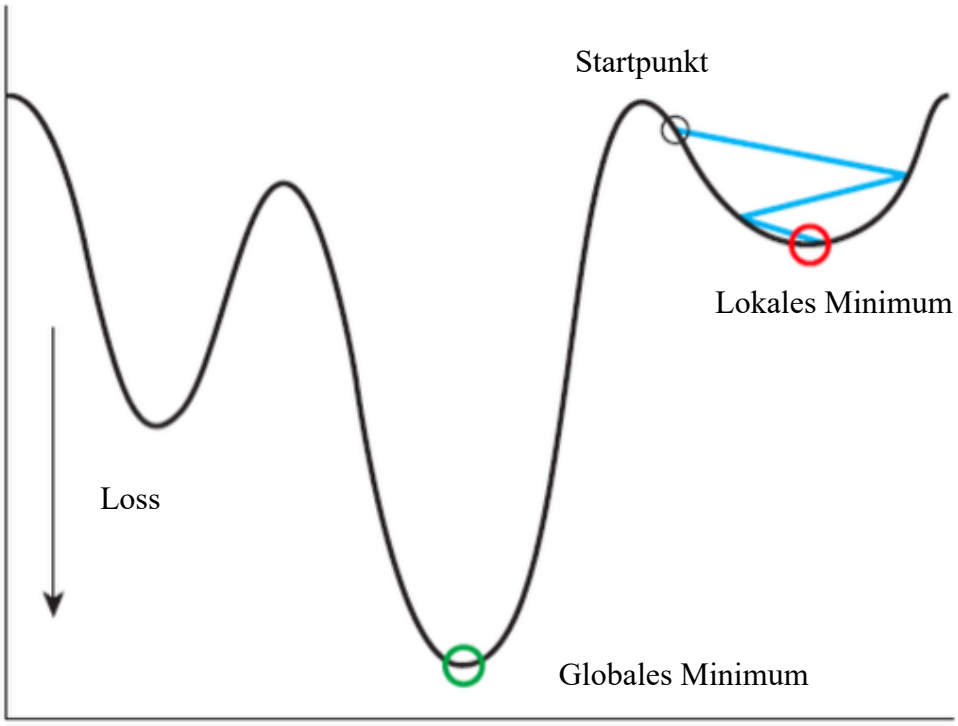
C.4.b

λ ist hierbei ein Hyperparameter, welcher die Wichtigkeit der Regularisierung beschreibt, ähnlich wie die Learningrate die Wichtigkeit der Backward Propagation symbolisiert.

Initialisierung nennt sich das Zuweisen von Werten an Elemente (in diesem Fall nur Weights und Biases) eines NN noch vor dem ersten Training. Obwohl bei diesem Prozess Zufallswerte genutzt werden, ist der Bereich, in welchem sich diese befinden für späteres Training wichtig. Ist der Bereich zu klein, wird der Gradient während des Trainings zu klein sein (auch *Vanishing Gradient* genannt), was dafür sorgt, dass Änderungen an den vorderen Layers exponentiell klein sein werden und für Ineffizienz sorgen. Ist der Bereich zu gross, geschieht genau das Umgekehrte (auch *Exploding Gradient* genannt), Änderungen in den vorderen Layers werden exponentiell gross sein und verhindern die Konvergenz des NN an das globale oder lokale Minimum.

Für das NN wurde die *He-Normal-Initialisierung* gewählt, da vorwiegend die Leaky ReLU-Aktivierungsfunktion innerhalb des NN genutzt wird. (Kumar, 2017)

Adaptive Learningrate beschreibt eine weitere Methode zur Optimierung. Wie im Kapitel zur Backward Propagation beschrieben, hängt die finale Genauigkeit eines NN stark von der gewählten Learningrate ab.



(Everdu, 2025)[eigene Bearbeitung]

Wie aus der Abbildung entnehmbar, ist keine einzelne Learningrate der Weg zum Erfolg, sondern viel eher ein Verlauf der Learningrate. In der Darstellung erkennt man, dass anfänglich eine höhere Learningrate erfordert wäre, um dem Tal des lokalen Minimums zu entfliehen und in die Region des globalen Minimums zu gelangen. Bleibt die Learningrate nach diesem Schritt jedoch weiterhin hoch, so geschieht, wie im Kapitel zur Backward Propagation erklärt, ein Herumspringen der Klassifikationsgenauigkeit um das erwünschte Minimum. Das heisst, eine kleinere Learningrate wäre nun wünschenswert.

Um dieses Problem zu lösen, gibt es die Methode der Adaptive Learningrate, die Learningrate mit der Zeit zu reduzieren. Es gibt auch hier wieder viele Variationen, wie man dies anstellen möchte, ob man die Änderung der Learningrate auf der vergangenen Zeit basiert oder auf dem *Momentum* des Trainings. Das Momentum des Trainings ist der Schnitt vergangener Gradienten und kann genutzt werden, um zu sehen, ob das NN mit dem Training an einem Zeitpunkt am Ziel vorbeischiesst, oder ob die Learningrate zu hoch ist.

D. Programm

D.1. Einschränkungen

Da aus Budgetgründen weder Zugriff auf riesige Rechenserver noch auf ein Forscher- und Zeichnerteam besteht, musste von Anfang an ein Überblick über Einschränkungen bezüglich dieses Projekts verschafft werden, da NN im Vergleich zu anderen Algorithmen überwiegend ressourcenlastig sind (Donges, 2023). Zu den Hauptlimitationen zählen:

1. Datensätze: NN benötigen ausgiebige Datensätze, in der Regel gilt: je mehr Daten, desto besser, da so präziser Muster extrahiert werden können. Aus diesem Grund wurde das EMNIST-Datenset für das NN ausgewählt, da nur schwer tausende von Bildern von Hand hätten selbst gezeichnet werden können.

2. *Computing Power*: Das Training von NN ist sehr rechenlastig und umfassend, wie an den Forward- und Backward Propagation-Formeln erkennbar. Aus dem Grund wird versucht, das NN so klein wie möglich zu halten, wodurch die Menge an Berechnungen für Forward- und Backward Propagation reduziert wird (in gewissen Fällen hat dies auch eigene Vorteile, wird in späteren Kapiteln weiter elaboriert).

   Ein weiterer Faktor, der zur NN-Grösse beiträgt, ist die Input-Grösse, von welcher direkt die Grösse des Input-Layers abhängt, weshalb die Bilder des EMNIST-Datensets herunterskaliert werden. Das sorgt dafür, dass weniger Pixel pro Bild vorhanden sind und daher an sich weniger Informationen für Berechnungen zur Verfügung stehen (dies kann auch wiederum seine technischen Vor- und Nachteile haben, welche in späteren Kapiteln erläutert werden).

3. Programmiersprachen: Schnellere Programmiersprachen wie *C++* oder *Rust* bieten zwar mehr Potential bezüglich des Einsparens von Zeit, sie sind dafür aber auch deutlich komplexer als beispielsweise Python und sie erweisen sich im Falle des Erklärens als hinderlich. (Yagmur, 2022)

   Aus dieser Basis wird Python angewendet, obwohl dafür möglicherweise Effizienz eingebüsst wird, da weiterhin das Erklären der Optimierung in dieser Arbeit vorgezogen wird.

4. Benutzerfreundlichkeit: Aus Komplexitäts- und Zeitgründen wird darauf verzichtet, das Programm spezifisch benutzerfreundlich zu gestalten, da nicht die Nutzung des Programmes, sondern das Programm als Veranschaulichung im Vordergrund steht.

D.2. Programmaufbau

Um ein Programm zur Handschriftenerkennung zu erstellen, gibt es ein endloses Spektrum an Ansatzmöglichkeiten, alle mit ihren eigenen Vor- und Nachteilen. Allein bezüglich der NN gibt es eine grosse Auswahl, unter anderem die bekanntesten Typen, FCNN, CNN und RNN, welche im Kapitel zu den NN bereits angesprochen wurden.

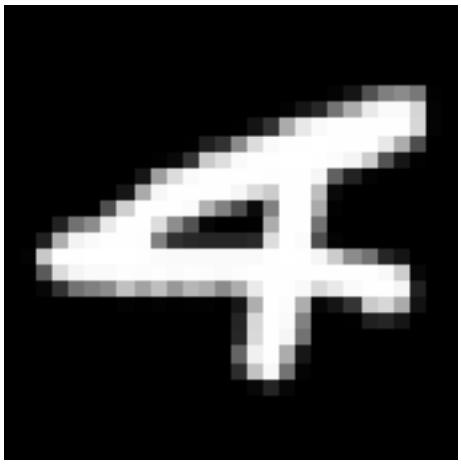
Aufgrund der bereits genannten Einschränkungen wird das NN an sich nur auf die einzelnen Buchstaben, statt auf die ganzen geschriebenen Zeilen trainiert. Das ist aufgrund der Druckschrift möglich, zumal ihre Buchstaben per Definition getrennt sein müssen und sich anhand dieser Voraussetzung ganze Zeilen mithilfe von einfachen Algorithmen in einzelne Buchstaben separieren lassen.

Die Nutzeffekte hiervon sprechen für sich: Es erfolgt eine allgemein deutlich erhöhte Effizienz und eine kleinere benötigte NN-Grösse sowie die Möglichkeit das NN-Modell direkt mit den EMNIST Daten zu trainieren, ohne diese zuerst zu unterschiedlichen Variationen von Zeilen zusammenfügen zu müssen.

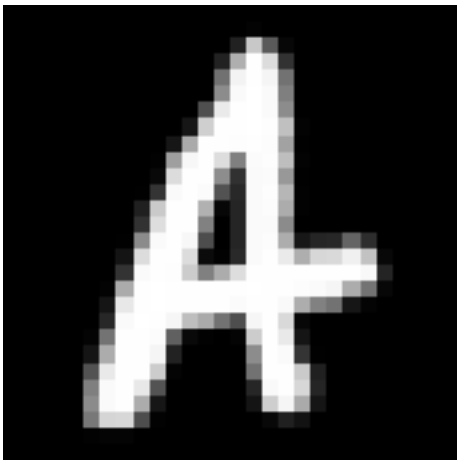
Da der Fokus auf dem NN selbst liegt, ist der Plan, dieses in den Mittelpunkt zu stellen. Dafür benötigt es einige Hilfsprogramme, welche die Daten so extrahieren und umformen, dass aus einer .png Datei eine Ansammlung an Werten wird, welche sich in den Input-Layer füttern lassen.

Um von den EMNIST Daten zu diesen besagten Werten zu kommen, braucht es unter anderem ein Programm, das die Daten herunterlädt, umformatiert, die Bilder von ihrem ursprünglichen 28×28 Pixel Format auf 8×8 Pixel herunterskaliert und aus designtechnischen Gründen (und als *Overflow*-Prävention, da die EMNIST Buchstaben ursprünglich weiss auf schwarzem Grund dargestellt, mehr Schwarz- als Weisswerte vorweisen würden und so zu hohe allgemeine Werte im NN auftauchen lassen würden) invertiert.

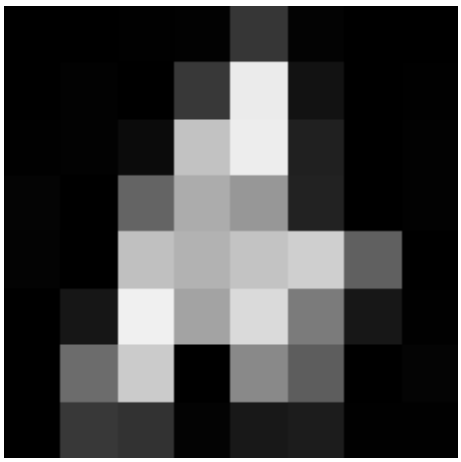
Nachdem die Pixelwerte auf den Bildern später mit einer MinMax-Normalisierung normalisiert wurden, kann mit diesen das NN trainiert werden. Hierbei ist die MinMax-Normalisierung eine Methode, mit welcher Werte linear auf einen gewünschten Bereich skaliert werden können. Diese ist in diesem Fall besonders praktisch, da sie Overflow vorbeugt und das NN flexibler gegenüber verschiedenen Datensätzen mit unterschiedlichen Wertebereichen macht.



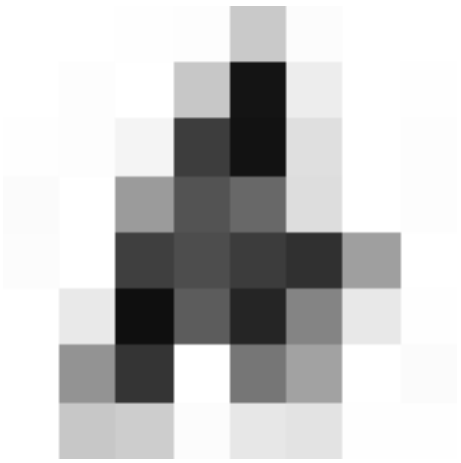
Unbearbeitetes Beispielbild aus dem EMNIST Dataset



Beispielbild gespiegelt und rotiert



Beispielbild von 28x28 Pixel auf 8x8 Pixel herunterskaliert



Beispielbild invertiert

255	255	254	253	201	252	255	255
255	253	255	199	20	237	255	254
254	253	244	61	18	223	255	253
251	255	155	83	104	221	255	253
252	255	63	77	60	48	159	255
255	233	15	92	37	132	232	254
255	147	52	255	118	162	255	251
255	199	205	252	231	227	254	254

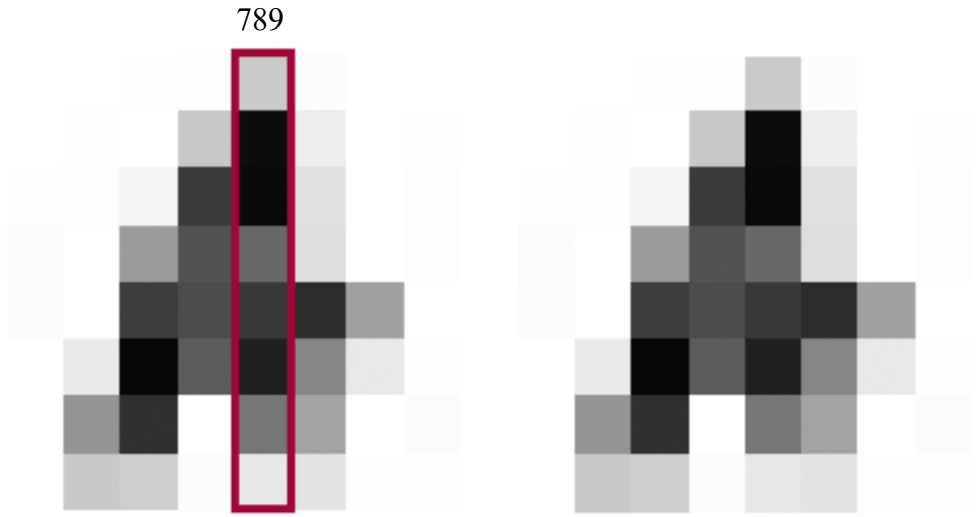
Beispielbild zu Arrays, welche dann in das NN gefüttert werden können

Dieser Teil des Programmes, die Bilder zu formatieren, wird jedoch in der Arbeit nicht noch weiter in der Tiefe erläutert, da er im Endeffekt für das eigentliche Training und auch für das Programm, weniger relevant ist. Er stellt nur eine Vorgehensweise dar, anstelle von manuellem Schreiben der erforderten Buchstaben die vorbereiteten des EMNIST zu verwenden.

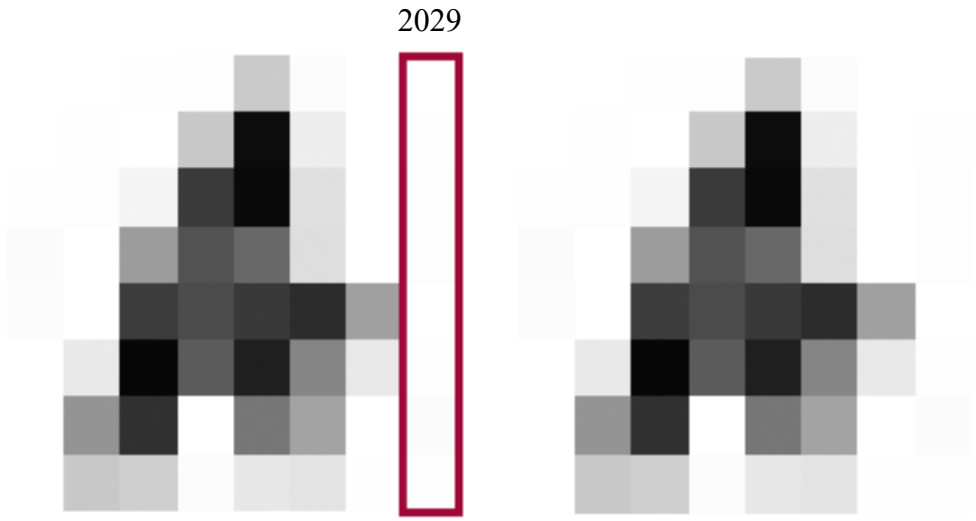
Für das endgültige Entziffern der ganzen Zeilen spaltet ein Algorithmus die Zeile in ihre einzelnen Buchstaben auf, indem fast leere, vertikale Linien gesucht werden.
Zwischengespeichert können dann diese Buchstabenabschnitte einzeln ins NN eingegeben werden. Die jeweiligen Outputs der Buchstaben werden zu einem Schlusssatz, also dem endgültigen Output zusammengefügt.

D.3. Programm

Am Anfang des Programmes steht der Spaltungsalgorithmus, der, wie bereits besprochen, Spalten an Pixel der Zeile addiert und die Unreinheit der Zeile (die Summe vom maximalen Schwarzwert der Zeile [die Zeilenhöhe mit 255, dem maximalen Pixelfarbwert bei Graustufen, multipliziert] subtrahiert) mit einem Hyperparameter vergleicht, sodass auch leere, aber leicht unreine Zeilen als Abstand erkannt werden.
Dieser Prozess wird für jede Spalte der Zeile an Buchstaben wiederholt.

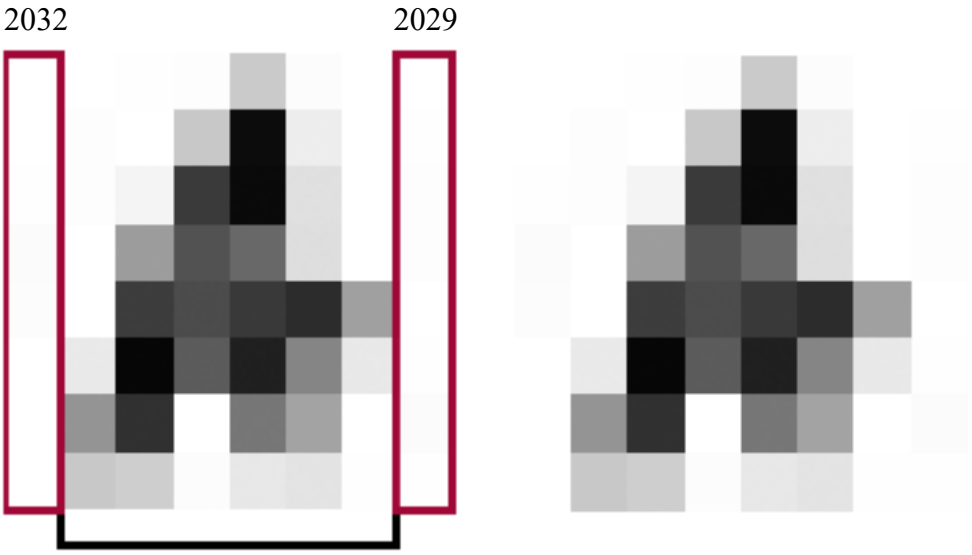


In diesem Beispiel, eine Zeile aus zweimal dem Buchstaben "A", ist die Summe der achten Spalte gleich 789. Diese Zahl wird von 2040 (8 Pixel, je maximal 255) subtrahiert, wodurch 1251 resultiert. Vergleicht man 1251 mit einem plausiblen Hyperparameter von 20, ergibt sich, dass die betroffene Spalte ziemlich unrein, also kein Leerraum, respektive Abstand, ist.

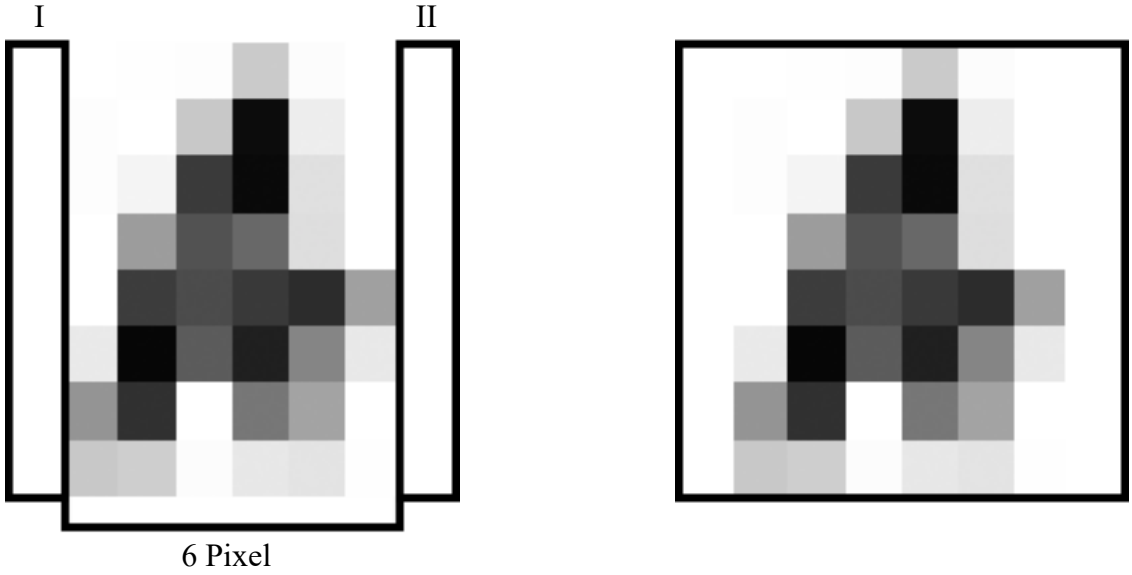


Bezieht man sich nun auf die Spalte 11 mit der Reinheit von 2029, beziehungsweise eine Unreinheit von 11, erschliesst sich, dass es sich hier tatsächlich um einen Abstand handelt.

Von Unreinheit, anstelle von der leichter zu berechnenden Summe ist hier die Rede, da sich mithilfe Ersterem leichter eine bessere Schätzung des Hyperparameters erlangen lässt und sich als Mensch Vorhandenes intuitiver als Abwesendes visualisieren lässt.



Beide Spalten, vor und nach dem ersten Buchstaben, werden als Abstand erkannt. Dies bedeutet für den Algorithmus, dass die Fläche zwischen den Abständen ein Buchstabe sein muss.



Fläche zwischen den Abständen auf weissem Canvas, seitlich je eine weisse Spalte

8 × 8 Pixel Bild mit dem extrahierten Buchstaben

Die Fläche wird anschliessend auf einen weissen 8 × 8 Canvas gelegt, sodass neben dem Buchstaben möglichst gleich viele Spalten an Weiss liegen und dieser ungefähr zentriert ist.

Für die Forward Propagation wurde eine simple und effiziente (in diesem Fall effizient, da eine externe Library für die Matrix-/Array-Berechnungen genutzt wird) Methode gewählt, welche mithilfe von Matrizen für eine saubere Berechnung sorgt.

Python - neuralnetwork.py

```
183 nodes = [zeros(layersize) for layersize in self.architecture]
184 nodes[0] = array(input) + self.model[0]["Biases"]
```

Es wird eine leere Liste fürs Zwischenspeichern der berechneten Werte erstellt. Die Input-Werte werden an erster Stelle der Liste mit den Biases verrechnet, indem die beiden Arrays, welche hier auch Matrizen sind, addiert werden.

Input-Layer Berechnung mit Matrizen, gefolgert von Gleichung C.1.f (im Fall des NN-Beispiels aus C.1):

$$\begin{bmatrix} Input_{1,1} \\ Input_{1,2} \end{bmatrix} + \begin{bmatrix} b_{1,1} \\ b_{1,2} \end{bmatrix} = \begin{bmatrix} z_{1,1} \\ z_{1,2} \end{bmatrix}$$

D.3.a

Python - neuralnetwork.py

```
187 for layer in range(1, self.modellen - 1):
188     prevactivations = applyactivation(nodes[layer - 1], False)
189     prevactivations = self.applydropout(prevactivations, self.dropoutrates[layer
    - 1], istraining)
190     nodes[layer] = dot(prevactivations, self.model[layer]["Weights"].T) +
    self.model[layer]["Biases"]
```

Die Forward Propagation wird von vorne nach hinten durchgeführt. Die zwischengespeicherten Werte des vorherigen Layers (z_l) werden je in die Aktivierungsfunktion eingegeben und als Liste erneut gespeichert. Anschliessend wird der Dropout angewendet, wenn das NN im Training ist, indem gewisse zwischengespeicherte Werte aus der neuen Liste gleich Null gesetzt werden. Danach werden die Werte des aktuellen Layers berechnet. Es gilt, von Gleichung C.1.e hergeleitet:

$$z_l = a_{l-1} \cdot (w_l)^T + b_l$$

D.3.b

Darstellung der Weights im Programm (im Fall des NN-Beispiels aus C.1):

$$\text{Weights} = \begin{bmatrix} w_{2,1,1} & w_{2,1,2} & w_{2,1,3} \\ w_{2,2,1} & w_{2,2,2} & w_{2,2,3} \end{bmatrix}, \begin{bmatrix} w_{3,1,1} & w_{3,1,2} & w_{3,1,3} \\ w_{3,2,1} & w_{3,2,2} & w_{3,2,3} \\ w_{3,3,1} & w_{3,3,2} & w_{3,3,3} \end{bmatrix}, \begin{bmatrix} w_{4,1,1} & \cdots \\ w_{4,2,1} & \cdots \\ w_{4,3,1} & \cdots \end{bmatrix}$$

D.3.c

Berechnung des zweiten Layers nach Gleichung D.3.b (im Fall des NN-Beispiels aus C.1):

$$\begin{bmatrix} a_{1,1} \\ a_{1,2} \end{bmatrix} = \phi\left(\begin{bmatrix} z_{1,1} \\ z_{1,2} \end{bmatrix}\right) = \begin{bmatrix} \phi(z_{1,1}) \\ \phi(z_{1,2}) \end{bmatrix}$$

D.3.d

$$\begin{bmatrix} z_{2,1} \\ z_{2,2} \\ z_{2,3} \end{bmatrix} = \begin{bmatrix} a_{1,1} \\ a_{1,2} \end{bmatrix} \cdot \begin{bmatrix} w_{2,1,1} & w_{2,2,1} \\ w_{2,1,2} & w_{2,2,2} \\ w_{2,1,3} & w_{2,2,3} \end{bmatrix} + \begin{bmatrix} b_{2,1} \\ b_{2,2} \\ b_{2,3} \end{bmatrix}$$

D.3.e

Python - neuralnetwork.py

```
192 finalactivations = applyactivation(nodes[-2], False)
193 finalactivations = self.applydropout(finalactivations, self.dropoutrates[-2],
    istraining)
```

Dass im vorletzten Layer separate Variablen gesetzt werden ist nun eigentlich überflüssig, doch boten sich in der Experimentierphase erweiterte Möglichkeiten zur Anpassung des NN.

Python - neuralnetwork.py
```
195 nodes[-1] = dot(finalactivations, self.model[-1]["Weights"].T) + self.model[-
    1]["Biases"]
196
197 return nodes
```

Im letzten Layer werden die Werte ebenfalls ohne Aktivierungsfunktion gespeichert, nur dass nun der Dropout nicht angewendet wird.

Die Backward Propagation wurde entgegen der Forward Propagation von hinten nach vorne, Layer für Layer, durchgeführt.

Python - neuralnetwork.py
```
126 errors = [array([0.0
127         for _ in self.model[layer]["Biases"]])
128         for layer in range(self.modellen - 1)]
129 + [activated_output - expected[inputid]]
```

Dieser Codeabschnitt zeigt die Erstellung einer Liste für das Zwischenspeichern von Neuron-Fehlern während der Backward Propagation.

In Zeile 129 werden gemäss Gleichung C.3.b die Fehler der Neuronen im Output-Layer gespeichert.

Python - neuralnetwork.py
```
136 nodederivative = applyderivative(output[layer][node],
137     layer == self.modellen - 1) * errors[layer][node]
138
139 delta = nodederivative * learningrate
```

In diesem Teil werden, aus Effizienz- und Darstellungsgründen, Berechnungen Variablen zugewiesen, da besagte Berechnungsresultate später noch mehrmals gebraucht werden.

Die Variablenwerte entsprechen der Gleichung C.3.i und dem Produkt von Gleichung C.3.i mit der Learningrate.

Python - neuralnetwork.py
```
142 modelcache[layer]["Biases"][node] -= delta
```

Der Bias wird nach Gleichung C.3.i angepasst.

Python - neuralnetwork.py
```
145 if layer != 0:
146     for prevnode in range(self.architecture[layer - 1]):
```

Anschliessend werden, falls der Layer nicht der Input-Layer ist, die Fehler der Neuronen des vorherigen Layers berechnet und die Weights korrigiert.

Python - neuralnetwork.py
```
148 l2penalty = l2lambda * self.model[layer]["Weights"][node][prevnode] /
    minibatchsize
```

Ein Faktor der Gleichung C.4.b wird im obigen Abschnitt kalkuliert.

Python - neuralnetwork.py
```
151 errors[layer - 1][prevnode] += self.model[layer]["Weights"][node][prevnode] *
    nodederivative
```

Um den Fehler der vorherigen Neuronen zu errechnen, wird hier das verbindende Weight (gemäss Gleichung C.3.j) mit Gleichung C.3.i multipliziert.

Das entstandene Produkt ist die Funktion der Summe in Gleichung C.3.l (respektive die Funktion der äusseren Summe in Gleichung C.3.m), wobei die Laufvariable besagter Summe in diesem Fall der Index des Neurons ist, für welchen die gesamte besprochene Berechnung, in einer Iteration, durchgeführt wird.

Das bedeutet, wird die Backpropagation für den ganzen Layer durchgeführt, summiert sich die obige Berechnung (die Multiplikation von Gleichungen C.3.i und C.3.j) wiederholt in der Liste für Neuronen-Fehler. Dadurch entspricht der gespeicherte Fehlerwert für ein Neuron in einem vorherigen Layer, nach vollständiger Berechnung des aktuellen Layers, exakt der Gleichung C.3.m.

Python - neuralnetwork.py
```
154 activation = applyactivation(output[layer - 1][prevnode], False)
155 modelcache[layer]["Weights"][node][prevnode] -= (activation * delta + l2penalty)
```

Hier wird das Weight nach Gleichung C.3.f korrigiert, durch Multiplikation von Gleichungen C.3.d und C.3.i (aus welchen Gleichung C.3.f besteht) und Addition der Regularisierung nach C.4.b.

(Der vollständige Code ist im Appendix 1 - Das Programm dargestellt)

D.4. Trainingsphase

Um das NN zu trainieren, wird eine Architektur mit 64 Input-Neuronen, 128 Neuronen im ersten Hidden-Layer, 64 Neuronen im zweiten Hidden-Layer und 26 Output-Neuronen gewählt.

$$[64, 128, 64, 26]$$

Es wird ein Dropout von 0.0 (0%) im Input-Layer, 0.25 (25%) im ersten Hidden-Layer, 0.1 (10%) im zweiten Hidden-Layer und 0.0 (0%) im Output-Layer ersehen.

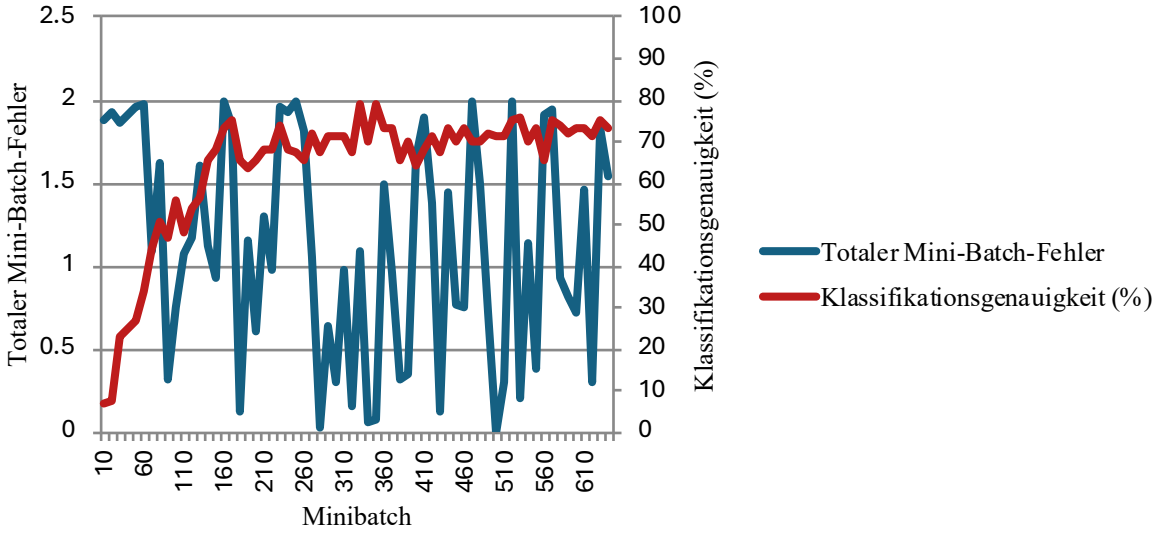
$$[0.0, 0.25, 0.1, 0.0]$$

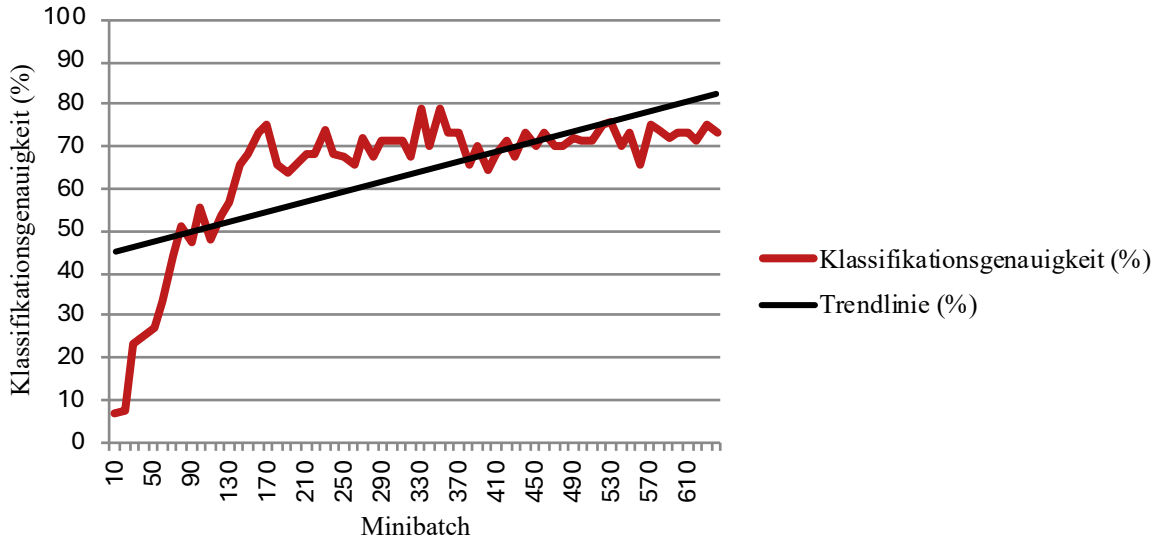
Ausserdem werden pro Buchstaben 3200 Trainingsdaten, mit einer Mini-Batch-Grösse von 128, $\eta = 0.003; \lambda = 0.00005$ genutzt.

Einer der grössten Faktoren in der Ausarbeitung der Hyperparameter war das Ausprobieren, da hierfür keine *one-fits-all-strategy* (engl.; Allzweckstrategie) existiert.

Für Messungen, beziehungsweise um sich während des Trainings einen Überblick zu verschaffen, wird nach jedem zehnten Mini-Batch der totale Fehler des Mini-Batches gespeichert, und das NN mit vier Testdaten (welche nicht unter den Trainingsdaten vorhanden sind) pro Buchstaben getestet.
Die vier Testdaten werden bei der Erstellung aller genutzter Daten zufällig festgelegt und bleiben während der verschiedenen Trainings konstant.

Die Resultate nach dem ersten Epoch des Trainings zeigen sich wie folgt:





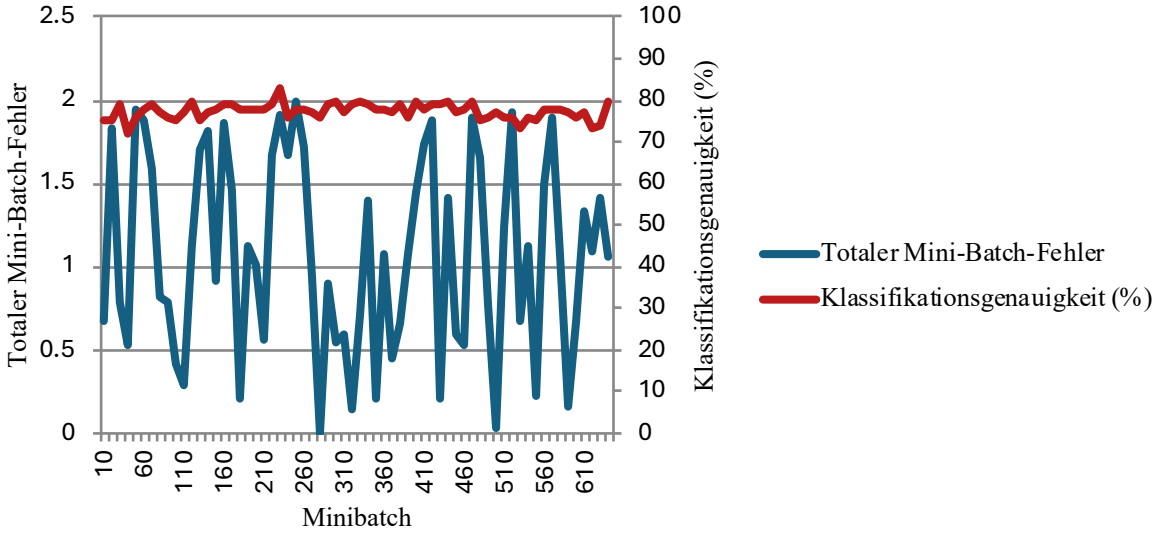
Stichproben-Varianz der Klassifikationsgenauigkeit:	257.36
Durchschnittlicher Totaler Mini-Batch-Fehler	1.13
Letzte Klassifikationsgenauigkeit in Prozent	73.1

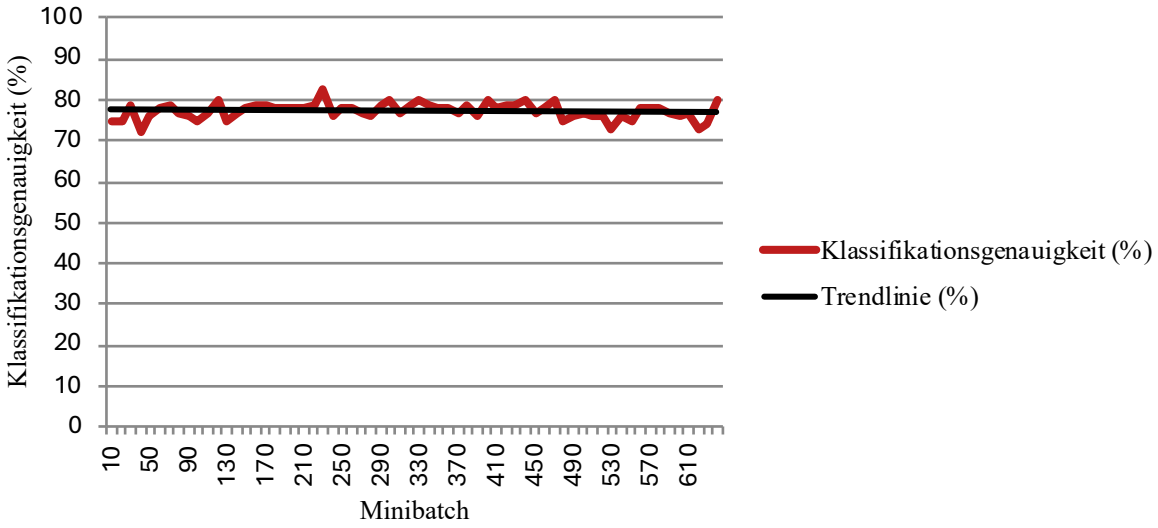
Resultate der Epoch 1

Nach der ersten Epoch (Testresultate sind genauer im Appendix 2 dargestellt) steht die Klassifikationsgenauigkeit bei 73.1 Prozent. Dies ist bereits ein ziemlich guter Wert, wenn man bedenkt, wie simpel das NN und wie komplex die Aufgabe ist.

Untersucht man die Einzelheiten des Testresultats, lassen sich leichte Muster bei den Fehlern des NN erkennen. Verwechselte Buchstaben sind oftmals dieselben und visuell ähnlich.

In der zweiten Epoch wird das NN mit $\eta = 0.0005; \lambda = 0.001$ trainiert.
Der Grund, wieso das λ nun so viel grösser als die Learningrate ist, ist das Ziel der Prävention von Overfitting. In den folgenden Epochs wird das NN Input-Werten ausgesetzt, welche schon mehrmals vorgekommen sind (im ersten Epoch und gegebenenfalls anderen vorherigen Epochs). Das riskiert Overfitting, gegen welches Regularisierung Aushilfe verschafft.
Die Resultate der zweiten Epoch sind folgende:



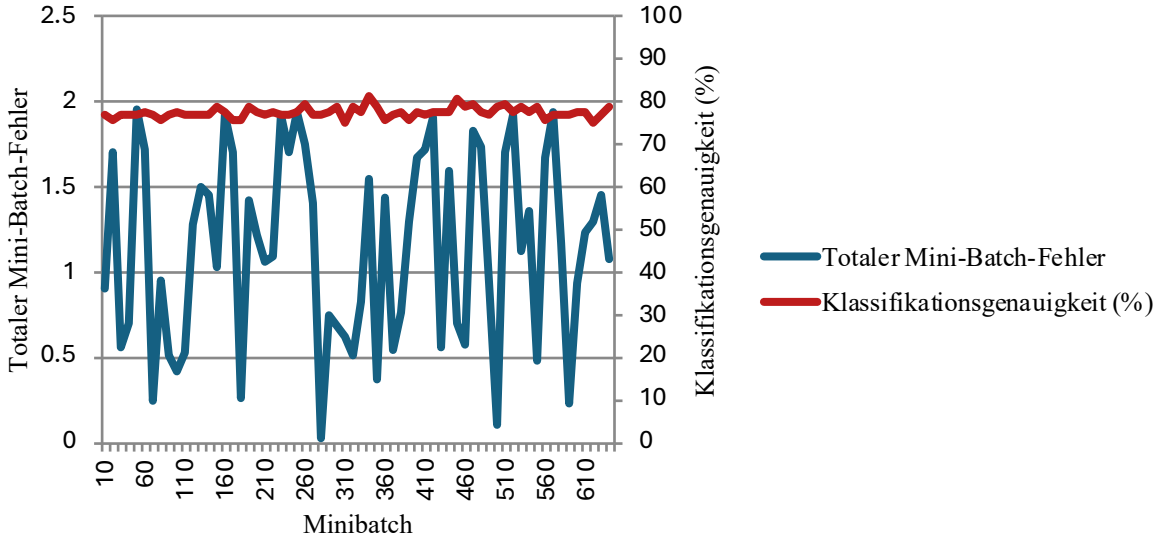


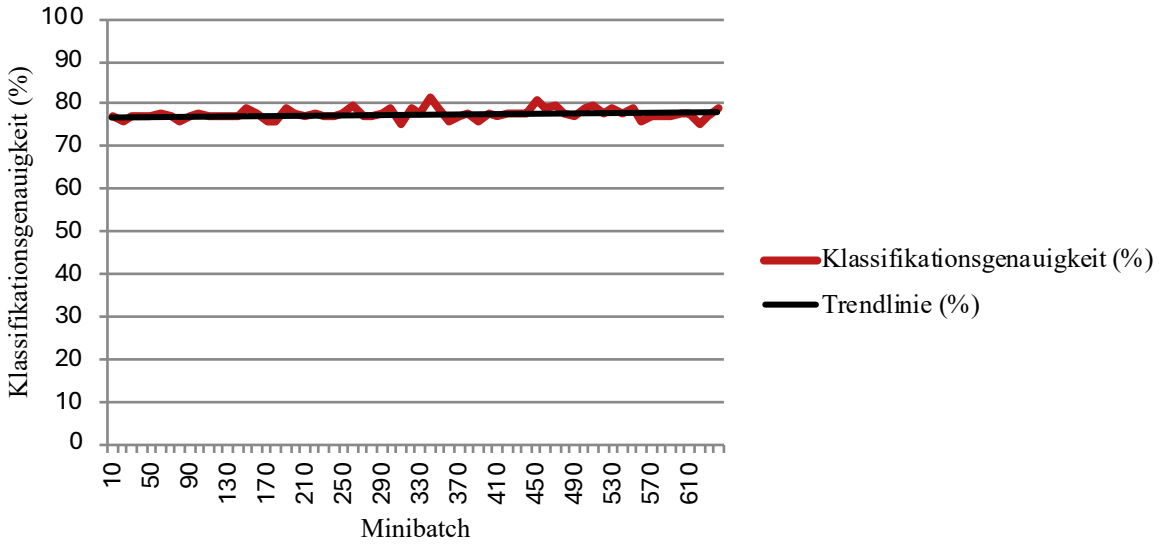
Stichproben-Varianz der Klassifikationsgenauigkeit:	3.60
Durchschnittlicher Totaler Mini-Batch-Fehler	1.07
Letzte Klassifikationsgenauigkeit in Prozent	77.9

Resultate der Epoch 2

Eine Testgenauigkeit von 77.9 Prozent nach der zweiten Epoch, eine erwartete Leistungssteigerung im Vergleich zur letzten.

In der dritten Epoch wird das NN mit $\eta = 0.0001; \lambda = 0.0003$ trainiert. Learningrate als auch λ sind nun noch tiefer als zuvor und sorgen so für feinere Verbesserungen:





Stichproben-Varianz der Klassifikationsgenauigkeit:	1.58
Durchschnittlicher Totaler Mini-Batch-Fehler	1.14
Letzte Klassifikationsgenauigkeit in Prozent	76.9

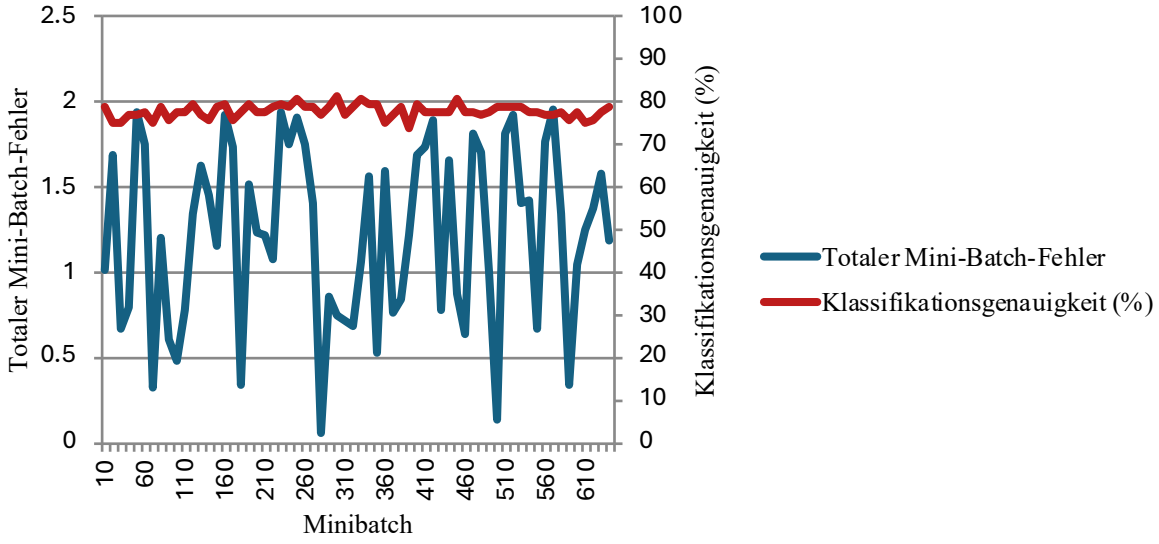
Resultate der Epoch 3

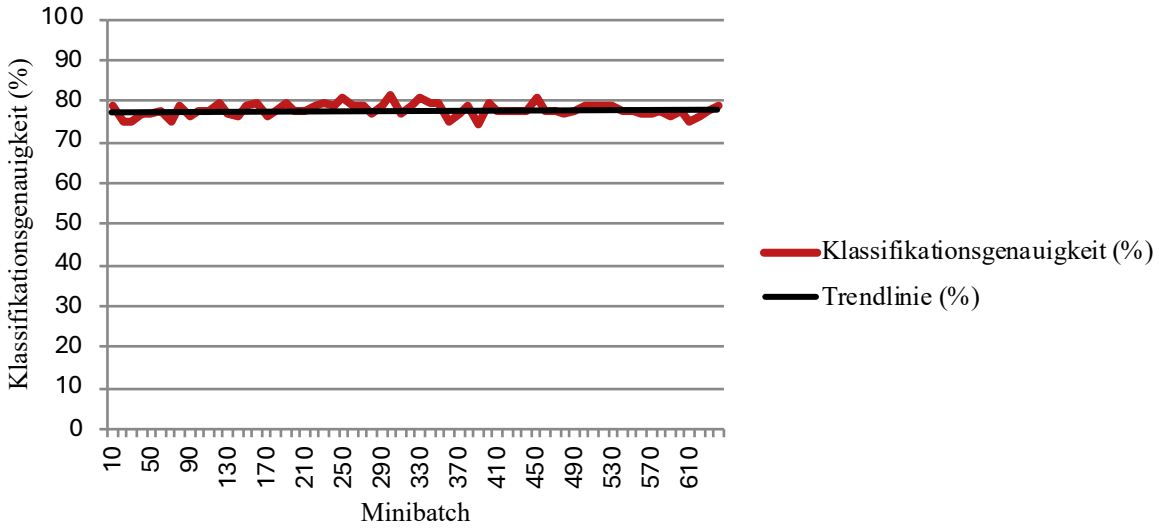
76.9 Prozent Genauigkeit, absolut gesehen zwar ein gutes Resultat, doch entspricht dies nicht den Erwartungen: Es liegt tiefer als in der zweiten Epoch.
Gründe dafür könnte es mehrere geben, zum Einen könnte es schlicht am Zufall liegen, dass das Training dieser Epoch an einer Stelle endete, an der die Performance des NN rein zufälligerweise an einem Tiefpunkt lag.
Zum anderen könnten leichtes Overfitting und Underfitting daran schuld sein, wenn die Balance zwischen L2-Regularisierung, Dropout und Learningrate nicht stimmte.

Die Kurve ist nun deutlich glatter, mit allgemein geringerer Varianz und einer recht abgeflachten Trendlinie, doch der durchschnittliche totale Mini-Batch-Fehler steigt. Es deutet stark darauf, dass sich das NN nun langsam in einem lokalen Minimum befindet, und die Sprünge aufgrund der kleineren Learningrate kleiner sind (Veranschaulicht im Bild zur Learningrate im Backward Propagation Kapitel). Noch besser wäre es, wenn es sich hierbei um

das globale Minimum handelt, weil dann mehr oder weniger das ganze Potenzial des NN ausgeschöpft wäre.

Für die vierte Epoch werden die gleichen Parameter wie für die dritte gewählt, für den letzten Feinschliff:





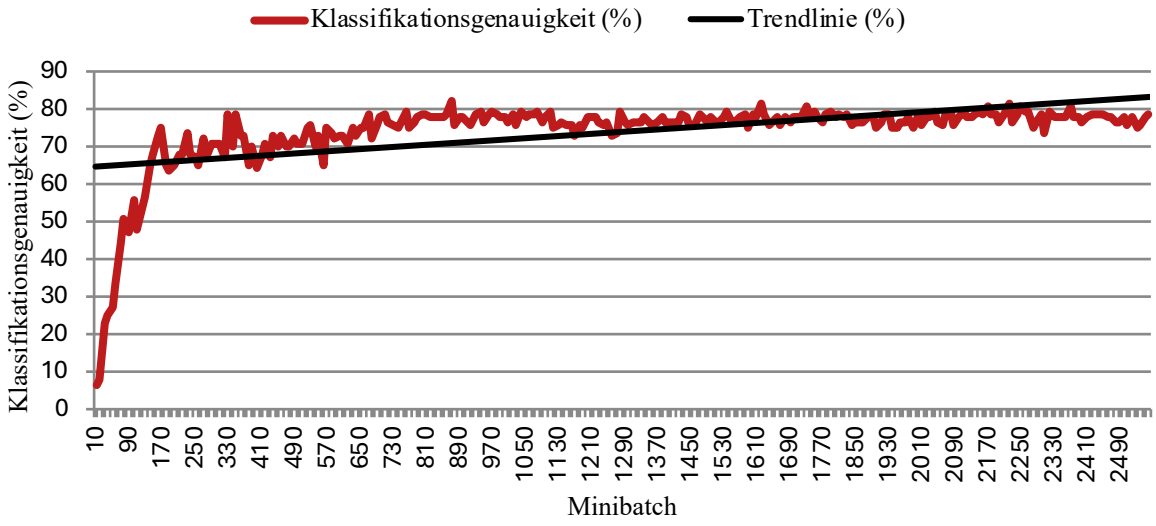
Stichproben-Varianz der Klassifikationsgenauigkeit:	2.56
Durchschnittlicher Totaler Mini-Batch-Fehler	1.22
Letzte Klassifikationsgenauigkeit in Prozent	78.8

Resultate der Epoch 4

78.8 Prozent, noch ein Stück besser als zuvor und es stimmt voll und ganz mit den Erwartungen eines kleinen Fortschritts überein.

Weiter wird das NN nicht mehr trainiert, da sich nach einigen Versuchen mit verschiedenen Hyperparametern nur noch Stagnation der Klassifikationsgenauigkeit feststellen liess, und das NN performancemässig für diese Arbeit bereits mehr als genügt.

In der Epoch 4 stieg sowohl die Varianz als auch der durchschnittliche totale Mini-Batch-Fehler, was auf eine Tendenz Richtung einer allgemeinen Verschlechterung der Performance weisen könnte.
Für die Stagnation finden sich ebenfalls mehr als genug Erklärungen, unter welchen die geringe Pixelanzahl mit zu tiefer Eindeutigkeit an Details von Buchstaben, als auch die kleine Grösse des NN am Wahrscheinlichsten empfunden werden können.



Stichproben-Varianz der Klassifikationsgenauigkeit: 101.22

Resultate des Trainings

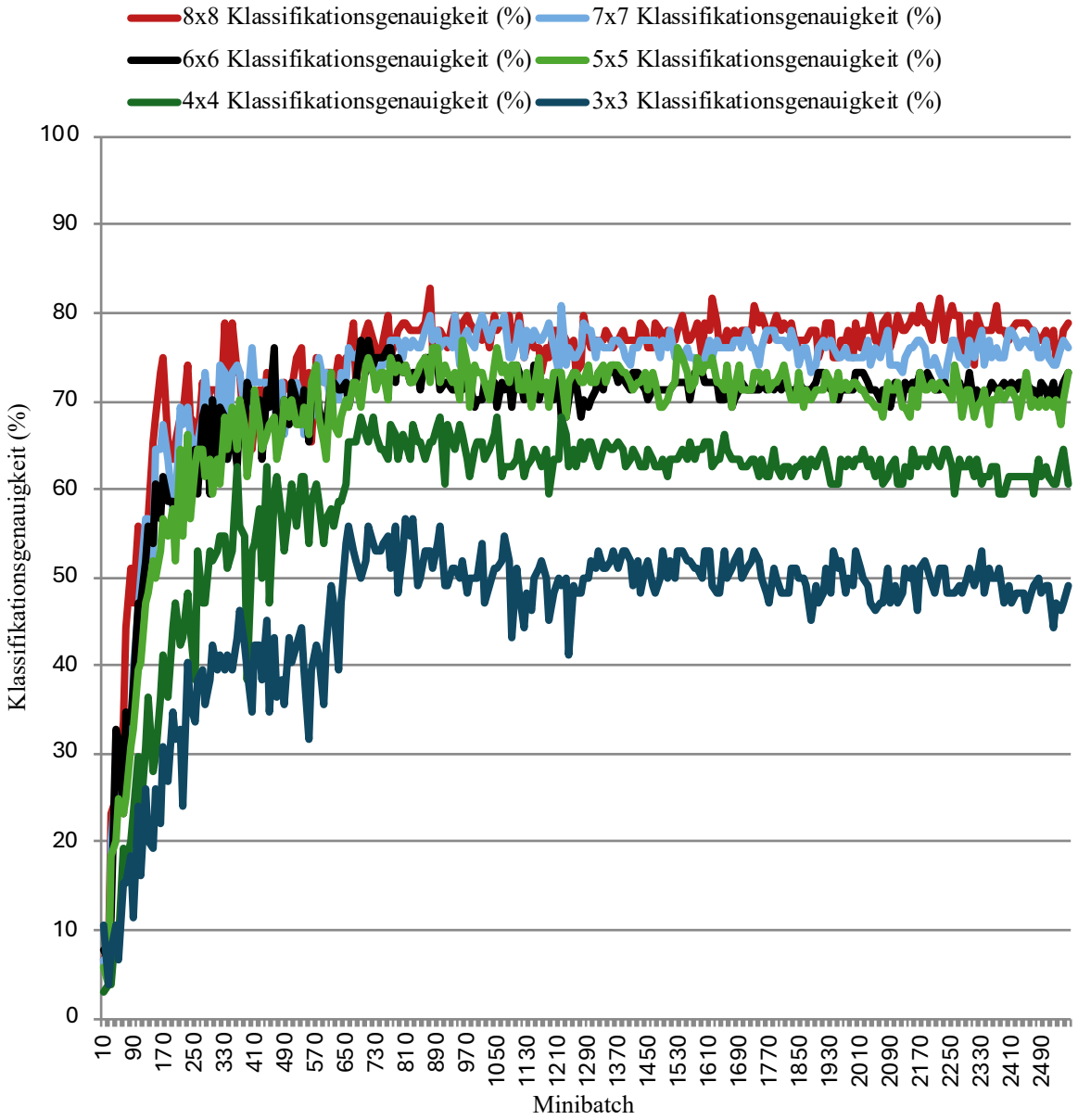
Analysiert man den totalen Verlauf der Klassifikationsgenauigkeit über das ganze Training, stellt man einen klaren Aufwärtstrend mit abflachender Tendenz fest.

D.5. Experiment mit kleineren Bildern

Um deutlich zu machen, welch eine wichtige Rolle die Input-Grösse im Training spielt, werden ähnliche NN (bis auf den Input-Layer identisch) mit 7×7, 6×6, 5×5, 4×4 und 3×3 anstelle von 8×8 Pixel Bildern, aber den gleichen Parametern trainiert.

Wegen den verschiedenen Bildgrössen wird bei der Architektur der NN jeweils die Input-Layer-Grösse auf die Anzahl Pixel pro Bild geändert, der Rest bleibt jedoch gleich.
Wissend, dass die gleichen Hyperparameter für eine kleinere Menge an Input-Werten zu nutzen ineffizient ist, dient dieses Experiment jedoch nur zu Demonstrationszwecken.
Hinzu kommt, dass die Auflösung bei weniger Pixeln deutlich schlechter ist, sodass in gewissen Fällen nicht einmal mehr Menschen korrekte Differenzierungen erkennen könnten.

Das Training, durchgeführt mit den kongruenten Parametern:

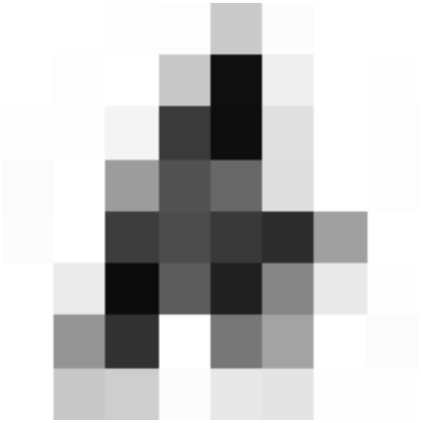


8 × 8 Finale Klassifikationsgenauigkeit in Prozent:	78.8
7 × 7 Finale Klassifikationsgenauigkeit in Prozent:	76.0
6 × 6 Finale Klassifikationsgenauigkeit in Prozent:	73.1
5 × 5 Finale Klassifikationsgenauigkeit in Prozent:	73.1
4 × 4 Finale Klassifikationsgenauigkeit in Prozent:	61.5
3 × 3 Finale Klassifikationsgenauigkeit in Prozent:	49.0

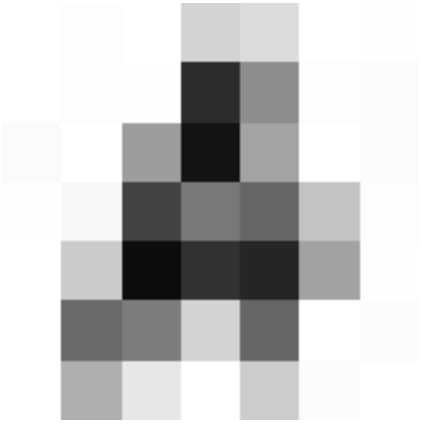
Vergleich der Trainings

Vergleicht man nun die finalen Klassifikationsgenauigkeiten der verschiedenen NN, so lässt sich klar ein Trend erkennen. Je höher die Bildauflösung, desto höher die finale Performance.

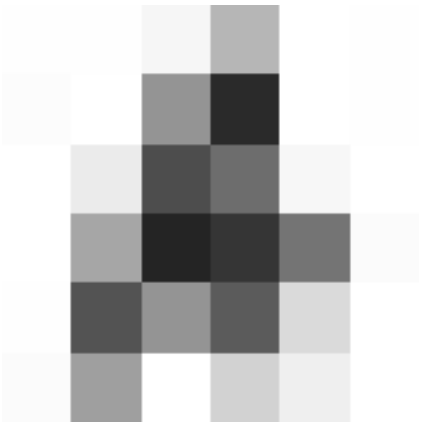
Bemerkenswert, dass das 3 × 3 NN entgegen den Erwartungen eine Genauigkeit von ganzen 49 Prozent erreichen konnte, bei einer Bildgrösse von insgesamt 9 Pixeln. Eine Errungenschaft, die für Menschen eigentlich unmöglich zu erlangen wäre.



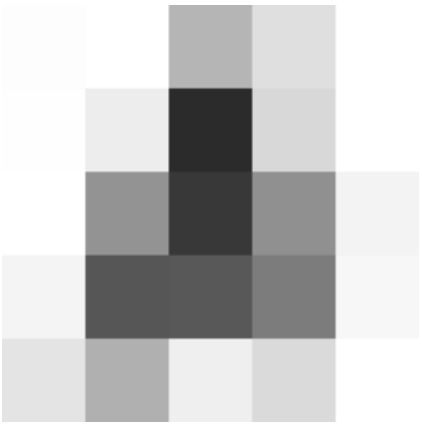
8 × 8 Bildgrösse → 64 Pixel



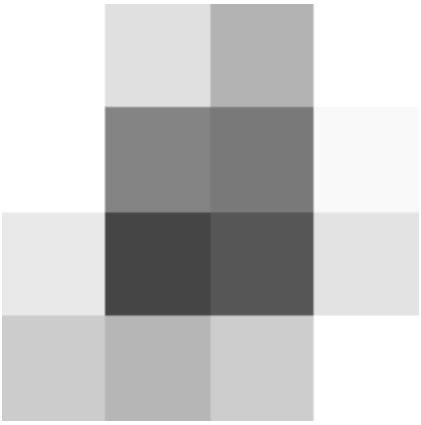
7 × 7 Bildgrösse → 49 Pixel



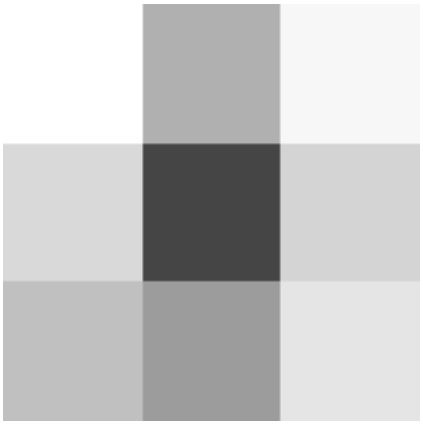
6 × 6 Bildgrösse → 36 Pixel



5 × 5 Bildgrösse → 25 Pixel



4 × 4 Bildgrösse → 16 Pixel



3 × 3 Bildgrösse → 9 Pixel

D.6. Konnex zur Handschriftenerkennung

Zusammenfassend lässt sich sagen, dass die Handschriftenerkennung ein sehr umfassendes und wechselhaftes Unterfangen ist. Keine zwei handgeschriebenen Buchstaben sind exakt gleich, geschweige denn ganze Handschriften verschiedener Menschen.
Es wäre unmöglich, Schrift zu entziffern, indem man die unendliche Anzahl aller geschriebenen Buchstaben auswendig lernt.

Für NN heisst dies, dass Overfitting unter allen Umständen vermieden werden muss. Das Erkennen von Buchstaben erfordert Generalisierung, gleichwohl genügend Kapazität, um die umfangreiche Anzahl an Buchstaben unterscheiden zu können.
Dies wurde im NN mithilfe von Dropout, L2-Regularisierung, der NN-Grösse an sich, wenigen, aber grossen Epochs und dem frühzeitigen Beenden des Trainings erreicht.

Zudem bedeutet die obige Schlussfolgerung einen Bedarf an möglichst vielen Daten für die Trainingsphase, aufgrund der hohen Variabilität an geschriebenen Buchstaben. Hier sorgte das EMNIST-Datenset für Abhilfe.

Jedoch ist die genutzte Methode für das Programm nur eine von vielen möglichen Ansätzen und hinsichtlich der technischen Einschränkungen leider nicht die effizienteste.

Wie besprochen, schreibt jeder Mensch Buchstaben auf seine eigene Art und Weise.
Dadurch ist es möglich, dass einzelne Buchstaben aus dem Kontext gerissen, wie sie das NN hier erkennen muss, gar auch von Menschen verwechselt werden können.

Das Gesamtbild aller Buchstaben, aus welchem ein Wort oder Satz resultiert, ist für uns Menschen und auch für NN ausschlaggebend. Ohne diesen Zusammenhang fehlt ein wichtiger Ansatzpunkt, um die Variabilität der menschlichen Handschrift zu kompensieren. Sprache und Verständnis sind Bestandteile des Lesens. (Learning, 2022)

Dem NN fehlt diese Möglichkeit leider, potenzielle Missverständnisse durch Verständnis des Geschriebenen zu verhindern.
Müsste also erneut ein Programm zur Handschriftenerkennung erstellt werden und ständen diesmal mehr Ressourcen zur Verfügung, wäre dies eine der wichtigsten Fähigkeiten, welche diesem Programm verliehen werden müsste. Um das zu erreichen, wäre ein RNN (oder noch besser ein *LLM*, *Large Language Model*) erfordert, welches kontextbezogene Schlüsse zwischen den potenziellen Buchstaben ziehen und so trotz einzelner Falschinterpretationen die richtige Endbotschaft erschliessen könnte.

Ein weiterer Punkt wäre, dass Buchstaben oftmals nur durch kleinere Details voneinander zu unterscheiden sind. Es ist manchmal nur ein kleiner Strich, der ein "I" zu einem "L" oder ein "F" zu einem "E" macht. Gehen diese Details verloren oder werden sie durch zufällige Schwankungen (in der Statistik auch *Noise* genannt) überschattet, werden korrekte Differenzierungen in der Schriftenerkennung unmöglich.
Dem entgegenzuwirken bedingt, dass diese wesentlichen Nuancen sichtbar bleiben oder sogar in den Vordergrund gerückt werden.
Möglichst grosse Bilder, in welchen diese Details kein Bestandteil eines Pixelwerts, sondern eigenständige Pixel sind, wären hier ein Weg der Bewerkstelligung (die Bildgrösse nicht zu verkleinern war in meinem Fall nur leider nicht möglich).

Pooling Layers aus CNN wären mithilfe von Maxima und Minima die perfekte Ergänzung dazu, da sie anschliessend markante Pixel, welche genau diese zentralen Feinheiten sind, herauspicken und hervorheben könnten.

D.7. Testphase

Zum Testen des trainierten NN wurden handgeschriebene Zeilen als Schrift gewählt, geschrieben von fünf verschiedenen Personen, um möglichst viel Variation abzudecken.

Test 1 - NEURAL NETWORK - geschrieben von Arne Viridén (KSL, Urdorf)

Handgeschriebene Zeile

Auf 8 Pixel Höhe herunterskaliert

Output - main.py

```
1  H
2  E
3  U
4  R
5  K
6  L
7  X
8  E
9  T
10 W
11 O
12 R
13 K
14 ['H', 'E', 'U', 'R', 'K', 'L', 'X', 'E', 'T', 'W', 'O', 'R', 'K']
```

Im ersten Test erzielte das NN eine Genauigkeit von 76.9 Prozent (10/13 Buchstaben). Die Hauptfehlerquellen sind die Buchstaben "N" und "A", genau wie in den Testresultaten der letzten Epoch ersichtlich.

Test 2 - COMPUTER SCIENCE - geschrieben von Mattia Cauterucci (KZU, Bülach)

Handgeschriebene Zeile

Auf 8 Pixel Höhe herunterskaliert

Output - main.py

```
C
J
M
P
V
T
F
F
I
C
I
F
N
C
F
['C', 'J', 'M', 'P', 'V', 'T', 'F', 'F', 'I', 'C', 'I', 'F', 'N', 'C', 'F']
```

Im zweiten Test erzielte das NN eine Genauigkeit von 53.3 Prozent (8/15 Buchstaben). Die Hauptfehlerquelle ist der Buchstabe "E", welcher in dieser Zeile gleich dreimal vorkommt. Verblüffend, da das NN alle vier Tests bezüglich des Buchstabens "E" im Klassifikationstest nach dem Training bestanden hatte.

Test 3 - GRADIENT DESCENT - geschrieben von Dominik Lee Friedrich (BZZ, Horgen)

Handgeschriebene Zeile

Auf 8 Pixel Höhe herunterskaliert

Output - main.py

```
G
K
K
D
I
E
H
T
D
E
S
C
B
B
T
['G', 'K', 'K', 'D', 'I', 'E', 'H', 'T', 'D', 'E', 'S', 'C', 'B', 'B', 'T']
```

Eine Genauigkeit von 66.7 Prozent (10/15 Buchstaben) wurde im dritten Test erreicht.

Test 4 - MACHINE LEARNING - geschrieben von Remy Bena (KBW, Büelrain)

Handgeschriebene Zeile

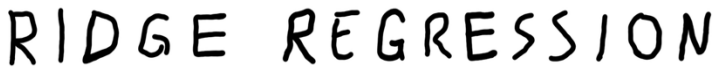
Auf 8 Pixel Höhe herunterskaliert

Output - main.py

```
1  M
2  K
3  C
4  H
5  I
6  E
7  E
8  V
9  E
10 F
11 K
12 E
13 I
14 V
15 G
16 ['M', 'K', 'C', 'H', 'I', 'E', 'E', 'V', 'E', 'F', 'K', 'E', 'I', 'V', 'G']
```

Im vierten Test erzielte das NN erneut eine Genauigkeit von 53.3 Prozent (8/15 Buchstaben).

Test 5 - RIDGE REGRESSION - geschrieben von Aydan Leuck (KSL, Urdorf)



Handgeschriebene Zeile

Auf 8 Pixel Höhe herunterskaliert

Output - main.py

```
Z
I
O
G
E
R
E
G
R
E
S
S
I
J
H
['Z', 'I', 'O', 'G', 'E', 'R', 'E', 'G', 'R', 'E', 'S', 'S', 'I', 'J', 'H']
```

Eine Genauigkeit von 73.3 Prozent (11/15 Buchstaben) im fünften Test.

D.8. Diskussion

In der Testphase traten nicht nur zufriedenstellende Resultate zutage, sondern zugleich manifestierten sich einige Muster.
Auffällig ist der Zusammenhang zwischen verwendeter Schriftdicke und der Genauigkeit wie auch das Verwechseln der Buchstaben, welche in den Klassifikationstests vom NN ebenfalls durcheinander gebracht wurden.

Besonders der Zusammenhang zwischen Klassifikationstest- und Testphasenergebnissen ist zwar markant, jedoch nicht allgemeingültig.
Also lässt sich daraus schliessen, dass der eingesetzte Klassifikationstest, mit welchem das NN jeweils nach einer Epoch getestet wurde, definitiv umfassender hätte sein können. Die erhöhte Variabilität der Buchstaben in der Testphase bewiesen, dass gutes Abschneiden in den vier Tests eines Buchstaben im Klassifikationstest nicht gleich das erfolgreiche Erkennen desselben Buchstabens in der fremden Zeile bedeutet.
Ein höheres Testvolumen hätte die umfangreiche Individualität von Handschrift besser abgedeckt und wäre ein besserer Prädiktor für die Performance des NN gewesen.

Vergleicht man das Klassifikationstestergebis und die Ergebnisse der Testphase, wirken diese nicht nur wie bereits angesprochen recht unterschiedlich und überraschend unerwartet, sondern sie weisen auch ein Muster auf. Eine Tendenz, bei der Testphase mit den Zeilen schlechter abzuschneiden als im isolierten Klassifikationstest des NN.
Es finden sich auch hier eine Vielzahl an Gründen, wobei der Zufall erneut eine prominente Rolle spielt. Der Test des NN basierte schlichtweg auf vier zufällig ausgewählten Inputs pro Buchstaben und die Zeilen aus zufälligen Konstellationen, die nicht gegenseitig gross miteinander zu tun haben müssen.

Doch auch als entscheidend für die verschlechterte Testphasenperformance kann die Schriftdicke, welche beim EMNIST eigentlich eher konstant war, bei den Zeilen aber weniger und die Buchstabenhäufigkeit der englischen Sprache gehalten werden (englische Sprache, da alle Zeilen englische Begriffe waren oder aus englischen Wörtern bestanden).
Die Schriftdicke kann natürlich wegen der nicht Linearität eines NN eine grosse Auswirkung auf den Output haben und diesen wesentlich verändern.
Die Buchstabenhäufigkeit der englischen Sprache beschreibt, dass Buchstaben wie "A", mit welchen das NN im Klassifikationstest Probleme hatte, oft, hingegen die drei seltensten Buchstaben, wie "J", "Q" und "X", die das NN fast perfekt erkannte, kaum vorkommen. (Raiders, 2023)
Dass diese drei Buchstaben am leichtesten erkennbar waren, lässt sich wahrscheinlich durch ihre eher einzigartige Form mit schwer verwechselbaren Elementen, respektive markanten Merkmalen erklären. Dass diese drei Buchstaben extrem selten in Wörtern vorkommen, ist rein zufallsbasiert.

Weitere, nicht zu vernachlässigende Komponenten sind die Art und Weise, wie die Zeilen auf die erwünschte Höhe skaliert wurden und der Algorithmus im Programm, welcher die Zeile anschliessend spaltet.
Im EMNIST wurden die ursprünglichen, handgeschriebenen Buchstaben mithilfe einer mehrstufigen Methode zentriert und anschliessend verkleinert. (NIST, 2024)
Im Prozess, welcher in dieser Arbeit genutzt wurde, wurde jedoch auf ein allzu kompliziertes Prinzip verzichtet, die Zeilen wurden ausschliesslich mit der "Bikubisch schärfer (Verkleinerung)" Funktion in Photoshop verkleinert und die Buchstaben auf der Zeile nicht absichtlich nachträglich zentriert.

Auch im Spaltungs-Algorithmus wurden die extrahierten Buchstaben gegebenenfalls nur grob mit weissen Spalten erweitert, was keineswegs ein richtiger Ersatz für Zentrierung ist.
Dieser Aspekt trägt auch dazu bei, dass die Leistung des NN in der Testphase im Vergleich zum Klassifizierungstest nachgelassen hat.

E. Reflexion

Im Laufe der Arbeit wurde mit jeder Menge an Variationen von NN und Hyperparametern herumprobiert, viel über die Algorithmen und den Calculus hinter NN herausgefunden und vielleicht die eine oder andere Lektion gelernt.

Schon früh wurde angefangen, sich über dieses Thema zu informieren, schon bevor festgelegt war, dass es zum Maturitätsarbeitsthema gemacht wird. Es erwies sich als äusserst nützlich, bereits eine Vorahnung vom Subjekt zu haben, besonders, wenn es ein dermassen komplexes, wie dieses, ist. Obendrein rückblickend eine Art Erkenntnis, Obliegenheiten mit hoher Priorität früh anzugehen, da diese, wie bei Mitschülern festgestellt, schnell zu Bürden werden können.

Angefangen wurde mit kleinen Testprogrammen, um sich an das Ganze heranzutasten. Dabei wurde von Anfang an auf allzu grosse Hilfestellungen oder Vorlagen von NN verzichtet. Eine Entscheidung, die kein Stück bereut wurde, da sie ein grundlegendes Verständnis für die Funktionsweisen und Algorithmen bis ins kleinste Detail bot. Die konstante Wiederkehr von gelernten Prinzipien und Konzepten bezüglich Calculus und NN liessen aufschliessen, wie wichtig die Aneignung der Bausteine zur Erleichterung in späteren Teilen der grösseren Arbeit ist.

Eine der wichtigsten Entscheidungen, die im Laufe der Arbeit getroffen wurde, war das EMNIST-Datenset zu nutzen. Als sich anfänglich ins Thema gestürzt wurde, war die ursprüngliche Idee, die eigene Handschrift zu nutzen. Jedoch kristallisierte sich über die Zeit heraus, wie wichtig ein hohes Volumen an Trainingsdaten ist, vor allem wenn es darum geht, 26 verschiedene Grossbuchstaben unseres Alphabets unterscheiden und kategorisieren zu können, wobei sich einige handgeschriebene Buchstaben gar von menschlicher Sicht aus recht schwierig unterscheiden lassen.

Das gleiche Muster, eine bestehende Methode völlig zu ersetzen und hinter sich lassen zu müssen, bewahrheitete sich im Allgemeinen als eine grosse Challenge. Unzählige ausgeklügelte Algorithmen und Programme mussten völlig aufgegeben werden, weil sich bessere, effizientere Lösungen ergaben. Die Einsicht, dass investierte Zeit verschwendet war und es einen besseren Weg gab, war, wie die *Sunk-Cost-Fallacy* (menschliche Tendenz, Vorhaben durchzuziehen, in welche schon viel investiert wurde, selbst wenn Aufgeben eindeutig die bessere Idee wäre) (MacNeil, 2025) beschreibt, sowohl eine Schwierigkeit, als auch ein wertvoller Lehrsatz, in der Zukunft bewusst auf Trugschlüsse zu achten.

Eine der ausdrücklichsten Ernüchterungen während des Herumexperimentierens war das Implementieren eines komplexen CNN-Algorithmus. Stereotypisch hat sich dieser für mein Projekt am vielversprechendsten angepriesen, zumal er gewissermassen als der Goldstandard der Bilderkennung gilt. Jedoch stellte sich am Ende heraus, dass die Bildgrösse 8×8 zu klein ist, um nach den Convolutions des CNN noch brauchbare Werte zurückzulassen. Eine höhere Bildgrösse stand aus technischen Gründen jedoch nicht im Raum, wodurch die Option eines CNN wegfiel.

Einen der herausforderndsten Teile dieser Arbeit verkörperte das Finetuning der Hyperparameter des NN. Es bieten sich unendlich viele Möglichkeiten, diese zu kombinieren, hingegen nur wenige genaue Richtlinien für deren Optimierung. Angesichts des nicht allzu schnellen Computers war das Durchprobieren verschiedenster Zusammenstellungen an Variablen und Parametern mithilfe von NN-Training ein zeitaufwändiger und herausfordernder Prozess. Ein Prozess, in welchem die virtuell unvorhersehbaren Schwankungen der

Klassifikationsgenauigkeiten und totalen Mini-Batch-Fehler regelrecht an Glücksspiel erinnerten, berauschender Nervenkitzel eines perfekt klassifizierten Mini-Batches und Niedergeschlagenheit auf Grund fallender Prozentzahlen.

Trotzdem gelang es nach stundenlangem Ausfertigen über mehrere Tage und Wochen dennoch, ein überraschend gutes NN zu trainieren, welches auch ziemlich zufriedenzustellen vermochte.

Hätte dieses Unterfangen mit dem nun gesammelten Hintergrundwissen angegangen werden können, würde definitiv eine Vergrösserung des Klassifikationstests an erster Stelle stehen, welcher mit seiner essentiellen Rolle als Rückmeldung zur Performance des NN einen direkten Einfluss auf dessen Anpassungen hat.
Ausserdem würde das Programm um ein Modul erweitert werden, welches die Fähigkeit besitzt den Output mithilfe eines Wörterbuches zu korrigieren. Grund dafür wäre die nicht zu vernachlässigende Bedeutung von Kontext im Thema der Schriftenerkennung.
Das Schlusslicht würden Anpassungen, wie ein noch grösserer Datensatz oder eine automatische Adaptive Learningrate, bilden.

Rückblickend lässt sich sagen, dass die Anfertigung meiner Maturitätsarbeit an sich wie auch die tiefgründige Recherche über ein hochspannendes Thema mit Zukunft sowie zahlreichen Anwendungen im echten Leben, wie beispielsweise als Analogie zur Neurologie (Subba Reddy Oota, 2023) und Psychologie (Levine, 1989), ein belohnendes und intellektuell weiterbildendes Unterfangen war.

F. Appendizes

F.1. Appendix 1 - Das Programm

Python - main.py

```python
import PIL
import PIL.Image
from numpy import *
import neuralnetwork
import functions

def getinput(folder, category, png):
    img = PIL.Image.open(folder+"/"+category+"/"+png)
    img.convert("L")
    imgwidth, imgheight = img.size

    #Gibt ein Array aus Arrays des Bildes zurück
    return array([array([float(sum(img.getpixel((x,y)))/3) for x in range(0,imgwidth)]) for y in range(0,imgheight)])

def splitinput(input, a=20):
    #Hyperparameter a bietet Spielraum für Unreinheiten zwischen Buchstaben

    sidelen, input, output, charstart = len(input), input.T, [], 0

    for i in range(len(input)):
        #Für jede Spalte des Bildes

        if ((255 * sidelen - input[i].sum()) < a):
            #Wenn die aktuelle Spalte als leer gilt, also ein Abstand ist

```

Python - main.py

```
            if ((i-charstart) < 2):
                #Wenn die Spalte kleiner gleich 1 Pixel ist, kein Buchstabe dazwischen passt
                charstart = i

            else:
                #else = wenn ein Abstand zwischen zwei Leerräumen erkannt wurde:

        margin = (sidelen - i + charstart + 1) / 2

                #Der Abschnitt wird zwischen zwei leere Weissflächen genommen, damit das Format 8x8 ist
                charpart = array([array([255] * sidelen)] * max(int(floor(margin)), 0) + [input[i]
                            for i in range(charstart+1, i)] + [array([255] * sidelen)] * max(int(ceil(margin)), 0)).T

                #Normalisiert den Abschnitt
                formattedcharpart = functions.minmaxnormalization(charpart, minmax=[0, 255]).flatten("F")

                #Forward Propagation mit dem Abschnitt als Input für das NN
                nnout = neuralnetwork.applyactivation(
                    neuralnetwork.main().run(formattedcharpart, istraining=False)[-1], True)

                #Deutet den NN Output, Kovertiert ihn zu einem Buchstaben
                charinterpretation = neuralnetwork.interpretcharacter(nnout)

                #Gibt den Output des NN für diesen Buchstaben aus
                print(charinterpretation)

                output += [charinterpretation]
                charstart = i

    #Gibt den Gesamten Output aus
    return print(output)
```

Python - neuralnetwork.py

```python
from sys import *; setrecursionlimit(500000)
from functions import *
import pngconverter
from numpy import *
from matplotlib.pyplot import *

#seed, damit Ergebnisse vergleichbar sind
random.seed(5)

leakyrelu = leakyreluactivation()

#Liste aus allen Grossbuchstaben
characters = [chr(i) for i in range(65, 91)]

#Dict, dass die Grossbuchstaben mit ihren gewünschten Outputs verbindet
#Outputs bestehen je aus 25 Nullen und einer Eins am Index des Grossbuchstaben
inoutdict = {characters[i]: array(i * [0] + [1] + (len(characters) - i - 1) * [0]) for i in range(len(characters))}

#Gibt den wahrscheinlichsten Buchstaben laut NN-Output aus
def interpretcharacter(neuralnetworkoutput):
    return characters[argmax(neuralnetworkoutput)]

#Funktion, die für Weights und Biases zufällige Werte ausgibt
#Nutzt He-Normal-Initialisierung
def randomvalue(target, fanin=1, fanout=1):
    randomranges = {"Weights": sqrt(2 / fanin), "Biases": 0.01}
    return random.uniform(-randomranges[target], randomranges[target])

#Wendet Aktivierungsfunktion je nach Layer an
def applyactivation(x, lastlayer: bool):
    return softmax(x) if lastlayer else leakyrelu.function(x)
```

Python - neuralnetwork.py

```
32
33 #Wendet Aktivierungsfunktionsderivative je nach Layer an
34 def applyderivative(x, lastlayer: bool):
35     return 1 if lastlayer else leakyrelu.derivative(x)
36
37 class NeuralNetwork():
38     def __init__(self, architecture, dropoutrates):
39         self.architecture = array([layer for layer in architecture])
40
41         #Speichert das NN
42         self.model = array([{
43                 "Weights": array([
44                     array([randomvalue("Weights", architecture[layer-1])
45                     for _ in range((architecture + [0])[layer-1])])
46                     for _ in range(architecture[layer])]),
47                 "Biases": array([randomvalue("Biases")
48                     for _ in range(architecture[layer])])}
49             for layer in range(len(architecture))])
50
51         #Anzahl Layer des NN als Konstante gespeichert, für Effizienz
52         self.modellen = len(self.model)
53
54         self.dropoutrates = dropoutrates
55
56     #Stellt die Werte der Elemente des NN dar
57     def printmodel(self):
58         for layer in range(self.modellen):
59             print(f"\nLayer {layer+1}:\n\nWeights:\n{
60                 self.model[layer]['Weights']}\n\nBiases:\n{
61                     self.model[layer]['Biases']}\n")
62
```

Python - neuralnetwork.py

```python
    #Speichert das NN in einer .npy (numpy) Datei
    def savemodel(self):
        with open("neuralnetwork.npy", "wb") as npyfile:
            save(npyfile, self.model)

    #Ruft das gespeicherte NN ab
    def getmodel(self):
        with open("neuralnetwork.npy", "rb") as npyfile:
            self.model = load(npyfile, allow_pickle=True)

    #Konvertiert, normalisiert, verbindet (Input mit jeweiligem Output) und speichert Daten
    def savedata(self, lort):
        if lort == "learn":
            learn = pngconverter.connect(inoutdict=inoutdict,folder="learn")
            with open("learndata.npy", "wb") as npyfile:
                save(npyfile, array(minmaxnormalization(learn["input"], minmax=[0, 255])))
                save(npyfile, array(learn["output"]))
        elif lort == "test":
            test = pngconverter.connect(inoutdict=inoutdict, folder="test")
            with open("testdata.npy", "wb") as npyfile:
                save(npyfile, array(minmaxnormalization(test["input"], minmax=[0, 255])))
                save(npyfile, array(test["output"]))

    #Ruft gespeicherte Daten ab
    def getdata(self, lort):
        if lort == "learn":
            with open("learndata.npy", "rb") as npyfile:
                return load(npyfile, allow_pickle=True), load(npyfile, allow_pickle=True)
        elif lort == "test":
            with open("testdata.npy", "rb") as npyfile:
                return load(npyfile, allow_pickle=True), load(npyfile, allow_pickle=True)
```

Python - neuralnetwork.py

```
94
95      #Funktion, die den Dropout anwendet
96      def applydropout(self, nodes, dropoutrate, istraining):
97          if not istraining or dropoutrate == 0:
98              return nodes
99          mask = random.binomial(1, 1 - dropoutrate, size=nodes.shape)
100         scale = 1.0 / (1.0 - dropoutrate)
101         return (nodes * mask) * scale
102
103     #Funktion, die die Backward Propagation anwendet
104     def train(self, input, expected, minibatchsize=128, epochs=1, learningrate=0.003, l2lambda=0.00005):
105
106         #seed wird gesetzt, damit Input und Output gleich gemischt werden
107         random.seed(6)
108         random.shuffle((input := input * epochs))
109         random.seed(6)
110         random.shuffle((expected := expected * epochs))
111
112         #Mini-Batch Gradient Descent
113         for minibatch in range(len(input)//minibatchsize):
114
115             #Kopie des NN
116             modelcache = self.model.copy()
117
118             for sample in range(minibatchsize):
119                 inputid = minibatch * minibatchsize + sample
120
121                 #Zwischenspeichert Forward Propagation Output
122                 output = self.run(input[inputid], istraining=True)
123                 activated_output = applyactivation(output[-1], True)
124
```

Python - neuralnetwork.py

```
125                 #Liste mit den Neuron-Fehlern
126                 errors = [array([0.0
127                         for _ in self.model[layer]["Biases"]])
128                         for layer in range(self.modellen - 1)]
129                 + [activated_output - expected[inputid]]
130 
131                 #Führt Backward Propagation von hinten nach vorne durch
132                 for layer in range(self.modellen - 1, -1, -1):
133                     for node in range(self.architecture[layer]):
134 
135                         #Backward Propagation für das Neuron
136                         nodederivative = applyderivative(output[layer][node],
137                             layer == self.modellen - 1) * errors[layer][node]
138 
139                         delta = nodederivative * learningrate
140 
141                         #Backward Propagation für den Bias
142                         modelcache[layer]["Biases"][node] -= delta
143 
144                         #Backward Propagation für Weights, wenn nicht Input-Layer
145                         if layer != 0:
146                             for prevnode in range(self.architecture[layer - 1]):
147                                 #L2-Regularisierung
148                                 l2penalty = l2lambda * self.model[layer]["Weights"][node][prevnode] / minibatchsize
149 
150                                 #Speichert den Neuron-Fehler des vorherigen Layers
151                                 errors[layer - 1][prevnode] += self.model[layer]["Weights"][node][prevnode] * nodederivative
152 
153                                 #Backward Propagation für das Weight
154                                 activation = applyactivation(output[layer - 1][prevnode], False)
155                                 modelcache[layer]["Weights"][node][prevnode] -= (activation * delta + l2penalty)
```

Python - neuralnetwork.py

```
156
157             #geändertes NN wird übernommen
158             self.model = modelcache
159
160     def test(self, input, expected):
161         correctoutput = 0
162         for i in range(len(input)):
163             output = self.run(input[i], istraining=False)[-1]
164             activated_output = applyactivation(output, True)
165
166             #Bool, ob der Output mit dem erwarteten Output übereinstimmt
167             outputbool = str(bool(interpretcharacter(expected[i]) == interpretcharacter(activated_output)))
168
169             correctoutput += int(outputbool == "True")
170             expectedchar, actualchar = interpretcharacter(expected[i]), interpretcharacter(activated_output)
171
172             print("Test {:{}} -> Expected: {:<{}} | Output: {:<{}} | Correct?: {:<{}}".format(
173                 str(i + 1), 3, expectedchar, 3, actualchar, 3, outputbool, 1))
174
175         #Gibt die Testgenauigkeit aus
176         print(f"results are {round(correctoutput / len(input) * 100, 1)} percent correct")
177
178     #Funktion, welche die Forward Propagation durchführt
179     def run(self, input, istraining=False):
180         #istraining Parameter, um zu bestimmen ob Dropout angewendet wird oder nicht
181
182         #Liste, in der Neuronwerte bei der Forward Propagation zwischengespeichert werden
183         nodes = [zeros(layersize) for layersize in self.architecture]
184         nodes[0] = array(input) + self.model[0]["Biases"]
185
```

Python - neuralnetwork.py

```
186         #Forward Propagation wird von vorne nach hinten durchgeführt
187         for layer in range(1, self.modellen - 1):
188             prevactivations = applyactivation(nodes[layer - 1], False)
189             prevactivations = self.applydropout(prevactivations, self.dropoutrates[layer - 1], istraining)
190             nodes[layer] = dot(prevactivations, self.model[layer]["Weights"].T) + self.model[layer]["Biases"]
191 
192         finalactivations = applyactivation(nodes[-2], False)
193         finalactivations = self.applydropout(finalactivations, self.dropoutrates[-2], istraining)
194 
195         nodes[-1] = dot(finalactivations, self.model[-1]["Weights"].T) + self.model[-1]["Biases"]
196 
197         return nodes
198 
199 def main():
200     global neuralnetwork
201     #Erstellt ein neues NN
202     neuralnetwork = NeuralNetwork(architecture=[64, 128, 64, 26], dropoutrates=[0.0, 0.25, 0.1, 0.0])
203 
204     #Fügt die gespeicherten Werte des NN ein
205     neuralnetwork.getmodel()
206     return neuralnetwork
207 
208 if __name__ == "__main__":
209     main()
210     neuralnetwork.savedata("learn")
211     neuralnetwork.savedata("test")
212     traindata = neuralnetwork.getdata("learn")
213     neuralnetwork.train(traindata[0], traindata[1])
214     testdata = neuralnetwork.getdata("test")
215     neuralnetwork.test(testdata[0], testdata[1])
216     neuralnetwork.savemodel()
```

Python - functions.py

```python
from numpy import *

class leakyreluactivation():
    def __init__(self, a=0.1):
        self.a = a

    def function(self, x):
        return maximum(x * self.a, x)

    def derivative(self, x):
        return 1 if x > 0 else self.a

def softmax(array):
    ex = sum([exp(x) for x in array])
    return [exp(x) / ex for x in array]

def meansquarederror(output, expected):
    if type(output) == type(int()) or type(output) == type(float()):
        return (output - expected) ** 2
    elif type(output) == type(list()):
        return sum([(output[i] - expected[i]) ** 2 for i in range(len(output))]) / len(output)

def meanerror(output, expected):
    if type(output) == type(int()) or type(output) == type(float()):
        return abs(output - expected)
    elif type(output) == type(list()):
        return sum([abs(output[i] - expected[i]) for i in range(len(output))]) / len(output)

```

Python - functions.py

```python
def minmaxnormalization(input, minmax=None, bounds=[0, 1]):
    #bounds Parameter setzt den Normalisierungs-Bereich
    #0 und 1 statt wie gewöhlich -1 und 1 gewählt, weil ich Leaky ReLU im NN nutze

    for argument in range(len(input[0])):

        if minmax == None:
            #findet den Minimal- und Maximalwert im Input, wenn nicht anders angegeben
            minmax = [min([element[argument] for element in input]), max([element[argument] for element in input])]

        for element in range(len(input)):
            try:
                input[element][argument] = nan_to_num((input[element][argument] - minmax[0]) / (
                    minmax[1] - minmax[0]) * (bounds[1] - bounds[0]) + bounds[0], nan=(
                    bounds[1] + bounds[0]) / 2, posinf=(bounds[1] + bounds[0]) / 2, neginf=(bounds[1] + bounds[0]) / 2)
            except ZeroDivisionError:
                input[element][argument] = (bounds[1] + bounds[0]) / 2

    return input
```

Python - pngconverter.py

```python
import os
import PIL
import PIL.Image
from numpy import *

def translate(folder):
    output = {}

    contents = os.listdir(folder)
    if ".DS_Store" in contents:
        #eine Datei die automatisch von Apple hinzugefügt wird, braucht man nicht
        contents.remove(".DS_Store")

    for category in contents:
        categoryoutput = []

        pngs = os.listdir(folder+"/"+category)
        if ".DS_Store" in pngs:
            pngs.remove(".DS_Store")

        for png in pngs:
            img = PIL.Image.open(folder+"/"+category+"/"+png)
            img.convert("RGB")
            imgwidth, imgheight = img.size

            #Wandelt das .png zu einem Input-Array für das NN um
            categoryoutput += [array([float(img.getpixel((x, y))) for x in range(0, imgwidth) for y in range(0, imgheight)])]

        output.update({category : categoryoutput})

    return output
```

Python - pngconverter.py

```
32
33 def connect(inoutdict={}, folder="learn"):
34     if inoutdict == {}:
35         raise RuntimeError
36
37     translation = translate(folder=folder)
38     outputin, outputout = [], []
39
40     for category in translation:
41         #Liste von Inputs für das NN
42         outputin += [png for png in translation[category]]
43
44         #Liste von Outputs für die dazugehörigen Inputs
```

F.2. Appendix 2 - Testergebnisse

Testergebnisse - Epoch 1 (Seite 1)

```
Test 1   -> Expected: R   | Output: X   | Correct?: False
Test 2   -> Expected: R   | Output: R   | Correct?: True
Test 3   -> Expected: R   | Output: B   | Correct?: False
Test 4   -> Expected: R   | Output: R   | Correct?: True
Test 5   -> Expected: U   | Output: U   | Correct?: True
Test 6   -> Expected: U   | Output: U   | Correct?: True
Test 7   -> Expected: U   | Output: U   | Correct?: True
Test 8   -> Expected: U   | Output: N   | Correct?: False
Test 9   -> Expected: I   | Output: I   | Correct?: True
Test 10  -> Expected: I   | Output: I   | Correct?: True
Test 11  -> Expected: I   | Output: I   | Correct?: True
Test 12  -> Expected: I   | Output: I   | Correct?: True
Test 13  -> Expected: N   | Output: N   | Correct?: True
Test 14  -> Expected: N   | Output: F   | Correct?: False
Test 15  -> Expected: N   | Output: H   | Correct?: False
Test 16  -> Expected: N   | Output: N   | Correct?: True
Test 17  -> Expected: G   | Output: G   | Correct?: True
Test 18  -> Expected: G   | Output: G   | Correct?: True
Test 19  -> Expected: G   | Output: G   | Correct?: True
Test 20  -> Expected: G   | Output: G   | Correct?: True
Test 21  -> Expected: Z   | Output: E   | Correct?: False
Test 22  -> Expected: Z   | Output: Z   | Correct?: True
Test 23  -> Expected: Z   | Output: Z   | Correct?: True
Test 24  -> Expected: Z   | Output: Z   | Correct?: True
Test 25  -> Expected: T   | Output: T   | Correct?: True
Test 26  -> Expected: T   | Output: M   | Correct?: False
Test 27  -> Expected: T   | Output: T   | Correct?: True
Test 28  -> Expected: T   | Output: T   | Correct?: True
Test 29  -> Expected: S   | Output: A   | Correct?: False
Test 30  -> Expected: S   | Output: S   | Correct?: True
Test 31  -> Expected: S   | Output: J   | Correct?: False
Test 32  -> Expected: S   | Output: G   | Correct?: False
Test 33  -> Expected: A   | Output: A   | Correct?: True
Test 34  -> Expected: A   | Output: A   | Correct?: True
Test 35  -> Expected: A   | Output: A   | Correct?: True
Test 36  -> Expected: A   | Output: A   | Correct?: True
Test 37  -> Expected: F   | Output: F   | Correct?: True
Test 38  -> Expected: F   | Output: F   | Correct?: True
Test 39  -> Expected: F   | Output: F   | Correct?: True
Test 40  -> Expected: F   | Output: F   | Correct?: True
Test 41  -> Expected: O   | Output: O   | Correct?: True
Test 42  -> Expected: O   | Output: D   | Correct?: False
Test 43  -> Expected: O   | Output: O   | Correct?: True
Test 44  -> Expected: O   | Output: O   | Correct?: True
Test 45  -> Expected: H   | Output: M   | Correct?: False
Test 46  -> Expected: H   | Output: A   | Correct?: False
Test 47  -> Expected: H   | Output: A   | Correct?: False
Test 48  -> Expected: H   | Output: A   | Correct?: False
Test 49  -> Expected: M   | Output: M   | Correct?: True
```

Testergebnisse - Epoch 1 (Seite 2)

```
Test 50  -> Expected: M   | Output: M   | Correct?: True
Test 51  -> Expected: M   | Output: M   | Correct?: True
Test 52  -> Expected: M   | Output: M   | Correct?: True
Test 53  -> Expected: J   | Output: J   | Correct?: True
Test 54  -> Expected: J   | Output: J   | Correct?: True
Test 55  -> Expected: J   | Output: J   | Correct?: True
Test 56  -> Expected: J   | Output: J   | Correct?: True
Test 57  -> Expected: C   | Output: C   | Correct?: True
Test 58  -> Expected: C   | Output: C   | Correct?: True
Test 59  -> Expected: C   | Output: L   | Correct?: False
Test 60  -> Expected: C   | Output: C   | Correct?: True
Test 61  -> Expected: D   | Output: D   | Correct?: True
Test 62  -> Expected: D   | Output: D   | Correct?: True
Test 63  -> Expected: D   | Output: O   | Correct?: False
Test 64  -> Expected: D   | Output: D   | Correct?: True
Test 65  -> Expected: V   | Output: V   | Correct?: True
Test 66  -> Expected: V   | Output: V   | Correct?: True
Test 67  -> Expected: V   | Output: V   | Correct?: True
Test 68  -> Expected: V   | Output: V   | Correct?: True
Test 69  -> Expected: Q   | Output: R   | Correct?: False
Test 70  -> Expected: Q   | Output: Q   | Correct?: True
Test 71  -> Expected: Q   | Output: C   | Correct?: False
Test 72  -> Expected: Q   | Output: G   | Correct?: False
Test 73  -> Expected: X   | Output: X   | Correct?: True
Test 74  -> Expected: X   | Output: X   | Correct?: True
Test 75  -> Expected: X   | Output: I   | Correct?: False
Test 76  -> Expected: X   | Output: X   | Correct?: True
Test 77  -> Expected: E   | Output: E   | Correct?: True
Test 78  -> Expected: E   | Output: E   | Correct?: True
Test 79  -> Expected: E   | Output: E   | Correct?: True
Test 80  -> Expected: E   | Output: E   | Correct?: True
Test 81  -> Expected: B   | Output: B   | Correct?: True
Test 82  -> Expected: B   | Output: D   | Correct?: False
Test 83  -> Expected: B   | Output: I   | Correct?: False
Test 84  -> Expected: B   | Output: O   | Correct?: False
Test 85  -> Expected: K   | Output: K   | Correct?: True
Test 86  -> Expected: K   | Output: R   | Correct?: False
Test 87  -> Expected: K   | Output: C   | Correct?: False
Test 88  -> Expected: K   | Output: K   | Correct?: True
Test 89  -> Expected: L   | Output: L   | Correct?: True
Test 90  -> Expected: L   | Output: L   | Correct?: True
Test 91  -> Expected: L   | Output: L   | Correct?: True
Test 92  -> Expected: L   | Output: L   | Correct?: True
Test 93  -> Expected: Y   | Output: Y   | Correct?: True
Test 94  -> Expected: Y   | Output: J   | Correct?: False
Test 95  -> Expected: Y   | Output: Y   | Correct?: True
Test 96  -> Expected: Y   | Output: Y   | Correct?: True
Test 97  -> Expected: P   | Output: Y   | Correct?: False
Test 98  -> Expected: P   | Output: P   | Correct?: True
Test 99  -> Expected: P   | Output: P   | Correct?: True
Test 100 -> Expected: P   | Output: P   | Correct?: True
```

Testergebnisse - Epoch 1 (Seite 3)

```
Test 101 -> Expected: W   | Output: W   | Correct?: True
Test 102 -> Expected: W   | Output: W   | Correct?: True
Test 103 -> Expected: W   | Output: W   | Correct?: True
Test 104 -> Expected: W   | Output: W   | Correct?: True
results are 73.1 percent correct
```

Testergebnisse - Epoch 2 (Seite 1)

```
Test 1   -> Expected: R   | Output: X   | Correct?: False
Test 2   -> Expected: R   | Output: R   | Correct?: True
Test 3   -> Expected: R   | Output: R   | Correct?: True
Test 4   -> Expected: R   | Output: R   | Correct?: True
Test 5   -> Expected: U   | Output: U   | Correct?: True
Test 6   -> Expected: U   | Output: U   | Correct?: True
Test 7   -> Expected: U   | Output: U   | Correct?: True
Test 8   -> Expected: U   | Output: U   | Correct?: True
Test 9   -> Expected: I   | Output: I   | Correct?: True
Test 10  -> Expected: I   | Output: I   | Correct?: True
Test 11  -> Expected: I   | Output: I   | Correct?: True
Test 12  -> Expected: I   | Output: I   | Correct?: True
Test 13  -> Expected: N   | Output: A   | Correct?: False
Test 14  -> Expected: N   | Output: A   | Correct?: False
Test 15  -> Expected: N   | Output: H   | Correct?: False
Test 16  -> Expected: N   | Output: N   | Correct?: True
Test 17  -> Expected: G   | Output: G   | Correct?: True
Test 18  -> Expected: G   | Output: G   | Correct?: True
Test 19  -> Expected: G   | Output: G   | Correct?: True
Test 20  -> Expected: G   | Output: G   | Correct?: True
Test 21  -> Expected: Z   | Output: B   | Correct?: False
Test 22  -> Expected: Z   | Output: Z   | Correct?: True
Test 23  -> Expected: Z   | Output: Z   | Correct?: True
Test 24  -> Expected: Z   | Output: Z   | Correct?: True
Test 25  -> Expected: T   | Output: T   | Correct?: True
Test 26  -> Expected: T   | Output: F   | Correct?: False
Test 27  -> Expected: T   | Output: T   | Correct?: True
Test 28  -> Expected: T   | Output: T   | Correct?: True
Test 29  -> Expected: S   | Output: A   | Correct?: False
Test 30  -> Expected: S   | Output: S   | Correct?: True
Test 31  -> Expected: S   | Output: S   | Correct?: True
Test 32  -> Expected: S   | Output: G   | Correct?: False
Test 33  -> Expected: A   | Output: A   | Correct?: True
Test 34  -> Expected: A   | Output: A   | Correct?: True
Test 35  -> Expected: A   | Output: A   | Correct?: True
Test 36  -> Expected: A   | Output: A   | Correct?: True
Test 37  -> Expected: F   | Output: F   | Correct?: True
Test 38  -> Expected: F   | Output: F   | Correct?: True
Test 39  -> Expected: F   | Output: F   | Correct?: True
Test 40  -> Expected: F   | Output: F   | Correct?: True
Test 41  -> Expected: O   | Output: O   | Correct?: True
Test 42  -> Expected: O   | Output: O   | Correct?: True
Test 43  -> Expected: O   | Output: O   | Correct?: True
Test 44  -> Expected: O   | Output: O   | Correct?: True
```

Testergebnisse - Epoch 2 (Seite 2)

```
Test 45  -> Expected: H   | Output: H   | Correct?: True
Test 46  -> Expected: H   | Output: A   | Correct?: False
Test 47  -> Expected: H   | Output: A   | Correct?: False
Test 48  -> Expected: H   | Output: H   | Correct?: True
Test 49  -> Expected: M   | Output: M   | Correct?: True
Test 50  -> Expected: M   | Output: M   | Correct?: True
Test 51  -> Expected: M   | Output: M   | Correct?: True
Test 52  -> Expected: M   | Output: M   | Correct?: True
Test 53  -> Expected: J   | Output: J   | Correct?: True
Test 54  -> Expected: J   | Output: J   | Correct?: True
Test 55  -> Expected: J   | Output: J   | Correct?: True
Test 56  -> Expected: J   | Output: J   | Correct?: True
Test 57  -> Expected: C   | Output: C   | Correct?: True
Test 58  -> Expected: C   | Output: C   | Correct?: True
Test 59  -> Expected: C   | Output: L   | Correct?: False
Test 60  -> Expected: C   | Output: C   | Correct?: True
Test 61  -> Expected: D   | Output: D   | Correct?: True
Test 62  -> Expected: D   | Output: D   | Correct?: True
Test 63  -> Expected: D   | Output: O   | Correct?: False
Test 64  -> Expected: D   | Output: D   | Correct?: True
Test 65  -> Expected: V   | Output: V   | Correct?: True
Test 66  -> Expected: V   | Output: V   | Correct?: True
Test 67  -> Expected: V   | Output: V   | Correct?: True
Test 68  -> Expected: V   | Output: V   | Correct?: True
Test 69  -> Expected: Q   | Output: R   | Correct?: False
Test 70  -> Expected: Q   | Output: Q   | Correct?: True
Test 71  -> Expected: Q   | Output: Q   | Correct?: True
Test 72  -> Expected: Q   | Output: Q   | Correct?: True
Test 73  -> Expected: X   | Output: X   | Correct?: True
Test 74  -> Expected: X   | Output: X   | Correct?: True
Test 75  -> Expected: X   | Output: I   | Correct?: False
Test 76  -> Expected: X   | Output: X   | Correct?: True
Test 77  -> Expected: E   | Output: E   | Correct?: True
Test 78  -> Expected: E   | Output: E   | Correct?: True
Test 79  -> Expected: E   | Output: E   | Correct?: True
Test 80  -> Expected: E   | Output: E   | Correct?: True
Test 81  -> Expected: B   | Output: D   | Correct?: False
Test 82  -> Expected: B   | Output: D   | Correct?: False
Test 83  -> Expected: B   | Output: F   | Correct?: False
Test 84  -> Expected: B   | Output: A   | Correct?: False
Test 85  -> Expected: K   | Output: K   | Correct?: True
Test 86  -> Expected: K   | Output: R   | Correct?: False
Test 87  -> Expected: K   | Output: K   | Correct?: True
Test 88  -> Expected: K   | Output: X   | Correct?: False
Test 89  -> Expected: L   | Output: L   | Correct?: True
Test 90  -> Expected: L   | Output: L   | Correct?: True
Test 91  -> Expected: L   | Output: L   | Correct?: True
Test 92  -> Expected: L   | Output: L   | Correct?: True
Test 93  -> Expected: Y   | Output: Y   | Correct?: True
Test 94  -> Expected: Y   | Output: J   | Correct?: False
Test 95  -> Expected: Y   | Output: Y   | Correct?: True
```

Testergebnisse - Epoch 2 (Seite 3)

```
Test 96  -> Expected: Y   | Output: P   | Correct?: False
Test 97  -> Expected: P   | Output: Y   | Correct?: False
Test 98  -> Expected: P   | Output: P   | Correct?: True
Test 99  -> Expected: P   | Output: P   | Correct?: True
Test 100 -> Expected: P   | Output: P   | Correct?: True
Test 101 -> Expected: W   | Output: W   | Correct?: True
Test 102 -> Expected: W   | Output: W   | Correct?: True
Test 103 -> Expected: W   | Output: W   | Correct?: True
Test 104 -> Expected: W   | Output: W   | Correct?: True
results are 77.9 percent correct
```

Testergebnisse - Epoch 3 (Seite 1)

```
Test 1   -> Expected: R   | Output: X   | Correct?: False
Test 2   -> Expected: R   | Output: R   | Correct?: True
Test 3   -> Expected: R   | Output: R   | Correct?: True
Test 4   -> Expected: R   | Output: R   | Correct?: True
Test 5   -> Expected: U   | Output: U   | Correct?: True
Test 6   -> Expected: U   | Output: U   | Correct?: True
Test 7   -> Expected: U   | Output: U   | Correct?: True
Test 8   -> Expected: U   | Output: U   | Correct?: True
Test 9   -> Expected: I   | Output: I   | Correct?: True
Test 10  -> Expected: I   | Output: I   | Correct?: True
Test 11  -> Expected: I   | Output: I   | Correct?: True
Test 12  -> Expected: I   | Output: I   | Correct?: True
Test 13  -> Expected: N   | Output: A   | Correct?: False
Test 14  -> Expected: N   | Output: F   | Correct?: False
Test 15  -> Expected: N   | Output: H   | Correct?: False
Test 16  -> Expected: N   | Output: N   | Correct?: True
Test 17  -> Expected: G   | Output: G   | Correct?: True
Test 18  -> Expected: G   | Output: G   | Correct?: True
Test 19  -> Expected: G   | Output: G   | Correct?: True
Test 20  -> Expected: G   | Output: G   | Correct?: True
Test 21  -> Expected: Z   | Output: B   | Correct?: False
Test 22  -> Expected: Z   | Output: Z   | Correct?: True
Test 23  -> Expected: Z   | Output: Z   | Correct?: True
Test 24  -> Expected: Z   | Output: Z   | Correct?: True
Test 25  -> Expected: T   | Output: T   | Correct?: True
Test 26  -> Expected: T   | Output: P   | Correct?: False
Test 27  -> Expected: T   | Output: T   | Correct?: True
Test 28  -> Expected: T   | Output: T   | Correct?: True
Test 29  -> Expected: S   | Output: A   | Correct?: False
Test 30  -> Expected: S   | Output: S   | Correct?: True
Test 31  -> Expected: S   | Output: S   | Correct?: True
Test 32  -> Expected: S   | Output: G   | Correct?: False
Test 33  -> Expected: A   | Output: M   | Correct?: False
Test 34  -> Expected: A   | Output: A   | Correct?: True
Test 35  -> Expected: A   | Output: A   | Correct?: True
Test 36  -> Expected: A   | Output: X   | Correct?: False
Test 37  -> Expected: F   | Output: F   | Correct?: True
Test 38  -> Expected: F   | Output: F   | Correct?: True
Test 39  -> Expected: F   | Output: F   | Correct?: True
```

Testergebnisse - Epoch 3 (Seite 2)

```
Test 40  -> Expected: F   | Output: F   | Correct?: True
Test 41  -> Expected: O   | Output: O   | Correct?: True
Test 42  -> Expected: O   | Output: O   | Correct?: True
Test 43  -> Expected: O   | Output: O   | Correct?: True
Test 44  -> Expected: O   | Output: O   | Correct?: True
Test 45  -> Expected: H   | Output: H   | Correct?: True
Test 46  -> Expected: H   | Output: A   | Correct?: False
Test 47  -> Expected: H   | Output: A   | Correct?: False
Test 48  -> Expected: H   | Output: H   | Correct?: True
Test 49  -> Expected: M   | Output: M   | Correct?: True
Test 50  -> Expected: M   | Output: M   | Correct?: True
Test 51  -> Expected: M   | Output: M   | Correct?: True
Test 52  -> Expected: M   | Output: M   | Correct?: True
Test 53  -> Expected: J   | Output: J   | Correct?: True
Test 54  -> Expected: J   | Output: J   | Correct?: True
Test 55  -> Expected: J   | Output: J   | Correct?: True
Test 56  -> Expected: J   | Output: J   | Correct?: True
Test 57  -> Expected: C   | Output: C   | Correct?: True
Test 58  -> Expected: C   | Output: C   | Correct?: True
Test 59  -> Expected: C   | Output: L   | Correct?: False
Test 60  -> Expected: C   | Output: C   | Correct?: True
Test 61  -> Expected: D   | Output: D   | Correct?: True
Test 62  -> Expected: D   | Output: D   | Correct?: True
Test 63  -> Expected: D   | Output: O   | Correct?: False
Test 64  -> Expected: D   | Output: D   | Correct?: True
Test 65  -> Expected: V   | Output: V   | Correct?: True
Test 66  -> Expected: V   | Output: V   | Correct?: True
Test 67  -> Expected: V   | Output: V   | Correct?: True
Test 68  -> Expected: V   | Output: V   | Correct?: True
Test 69  -> Expected: Q   | Output: R   | Correct?: False
Test 70  -> Expected: Q   | Output: Q   | Correct?: True
Test 71  -> Expected: Q   | Output: Q   | Correct?: True
Test 72  -> Expected: Q   | Output: Q   | Correct?: True
Test 73  -> Expected: X   | Output: X   | Correct?: True
Test 74  -> Expected: X   | Output: X   | Correct?: True
Test 75  -> Expected: X   | Output: I   | Correct?: False
Test 76  -> Expected: X   | Output: X   | Correct?: True
Test 77  -> Expected: E   | Output: E   | Correct?: True
Test 78  -> Expected: E   | Output: E   | Correct?: True
Test 79  -> Expected: E   | Output: E   | Correct?: True
Test 80  -> Expected: E   | Output: E   | Correct?: True
Test 81  -> Expected: B   | Output: D   | Correct?: False
Test 82  -> Expected: B   | Output: D   | Correct?: False
Test 83  -> Expected: B   | Output: A   | Correct?: False
Test 84  -> Expected: B   | Output: O   | Correct?: False
Test 85  -> Expected: K   | Output: K   | Correct?: True
Test 86  -> Expected: K   | Output: R   | Correct?: False
Test 87  -> Expected: K   | Output: K   | Correct?: True
Test 88  -> Expected: K   | Output: X   | Correct?: False
Test 89  -> Expected: L   | Output: L   | Correct?: True
Test 90  -> Expected: L   | Output: L   | Correct?: True
```

Testergebnisse - Epoch 3 (Seite 3)

```
Test 91  -> Expected: L   | Output: L   | Correct?: True
Test 92  -> Expected: L   | Output: L   | Correct?: True
Test 93  -> Expected: Y   | Output: Y   | Correct?: True
Test 94  -> Expected: Y   | Output: J   | Correct?: False
Test 95  -> Expected: Y   | Output: Y   | Correct?: True
Test 96  -> Expected: Y   | Output: P   | Correct?: False
Test 97  -> Expected: P   | Output: P   | Correct?: True
Test 98  -> Expected: P   | Output: P   | Correct?: True
Test 99  -> Expected: P   | Output: P   | Correct?: True
Test 100 -> Expected: P   | Output: P   | Correct?: True
Test 101 -> Expected: W   | Output: W   | Correct?: True
Test 102 -> Expected: W   | Output: W   | Correct?: True
Test 103 -> Expected: W   | Output: W   | Correct?: True
Test 104 -> Expected: W   | Output: W   | Correct?: True
results are 76.9 percent correct
```

Testergebnisse - Epoch 4 (Seite 1)

```
Test 1   -> Expected: R   | Output: X   | Correct?: False
Test 2   -> Expected: R   | Output: R   | Correct?: True
Test 3   -> Expected: R   | Output: R   | Correct?: True
Test 4   -> Expected: R   | Output: R   | Correct?: True
Test 5   -> Expected: U   | Output: U   | Correct?: True
Test 6   -> Expected: U   | Output: U   | Correct?: True
Test 7   -> Expected: U   | Output: U   | Correct?: True
Test 8   -> Expected: U   | Output: U   | Correct?: True
Test 9   -> Expected: I   | Output: I   | Correct?: True
Test 10  -> Expected: I   | Output: I   | Correct?: True
Test 11  -> Expected: I   | Output: I   | Correct?: True
Test 12  -> Expected: I   | Output: I   | Correct?: True
Test 13  -> Expected: N   | Output: N   | Correct?: True
Test 14  -> Expected: N   | Output: F   | Correct?: False
Test 15  -> Expected: N   | Output: H   | Correct?: False
Test 16  -> Expected: N   | Output: N   | Correct?: True
Test 17  -> Expected: G   | Output: G   | Correct?: True
Test 18  -> Expected: G   | Output: G   | Correct?: True
Test 19  -> Expected: G   | Output: G   | Correct?: True
Test 20  -> Expected: G   | Output: G   | Correct?: True
Test 21  -> Expected: Z   | Output: B   | Correct?: False
Test 22  -> Expected: Z   | Output: Z   | Correct?: True
Test 23  -> Expected: Z   | Output: Z   | Correct?: True
Test 24  -> Expected: Z   | Output: Z   | Correct?: True
Test 25  -> Expected: T   | Output: T   | Correct?: True
Test 26  -> Expected: T   | Output: P   | Correct?: False
Test 27  -> Expected: T   | Output: T   | Correct?: True
Test 28  -> Expected: T   | Output: T   | Correct?: True
Test 29  -> Expected: S   | Output: Y   | Correct?: False
Test 30  -> Expected: S   | Output: S   | Correct?: True
Test 31  -> Expected: S   | Output: S   | Correct?: True
Test 32  -> Expected: S   | Output: G   | Correct?: False
Test 33  -> Expected: A   | Output: M   | Correct?: False
Test 34  -> Expected: A   | Output: A   | Correct?: True
```

Testergebnisse - Epoch 4 (Seite 2)

```
Test 35  -> Expected: A   | Output: A   | Correct?: True
Test 36  -> Expected: A   | Output: X   | Correct?: False
Test 37  -> Expected: F   | Output: F   | Correct?: True
Test 38  -> Expected: F   | Output: F   | Correct?: True
Test 39  -> Expected: F   | Output: F   | Correct?: True
Test 40  -> Expected: F   | Output: F   | Correct?: True
Test 41  -> Expected: O   | Output: O   | Correct?: True
Test 42  -> Expected: O   | Output: O   | Correct?: True
Test 43  -> Expected: O   | Output: O   | Correct?: True
Test 44  -> Expected: O   | Output: O   | Correct?: True
Test 45  -> Expected: H   | Output: H   | Correct?: True
Test 46  -> Expected: H   | Output: A   | Correct?: False
Test 47  -> Expected: H   | Output: A   | Correct?: False
Test 48  -> Expected: H   | Output: H   | Correct?: True
Test 49  -> Expected: M   | Output: M   | Correct?: True
Test 50  -> Expected: M   | Output: M   | Correct?: True
Test 51  -> Expected: M   | Output: M   | Correct?: True
Test 52  -> Expected: M   | Output: M   | Correct?: True
Test 53  -> Expected: J   | Output: J   | Correct?: True
Test 54  -> Expected: J   | Output: J   | Correct?: True
Test 55  -> Expected: J   | Output: J   | Correct?: True
Test 56  -> Expected: J   | Output: J   | Correct?: True
Test 57  -> Expected: C   | Output: C   | Correct?: True
Test 58  -> Expected: C   | Output: C   | Correct?: True
Test 59  -> Expected: C   | Output: L   | Correct?: False
Test 60  -> Expected: C   | Output: C   | Correct?: True
Test 61  -> Expected: D   | Output: D   | Correct?: True
Test 62  -> Expected: D   | Output: D   | Correct?: True
Test 63  -> Expected: D   | Output: O   | Correct?: False
Test 64  -> Expected: D   | Output: D   | Correct?: True
Test 65  -> Expected: V   | Output: V   | Correct?: True
Test 66  -> Expected: V   | Output: V   | Correct?: True
Test 67  -> Expected: V   | Output: V   | Correct?: True
Test 68  -> Expected: V   | Output: V   | Correct?: True
Test 69  -> Expected: Q   | Output: R   | Correct?: False
Test 70  -> Expected: Q   | Output: Q   | Correct?: True
Test 71  -> Expected: Q   | Output: Q   | Correct?: True
Test 72  -> Expected: Q   | Output: Q   | Correct?: True
Test 73  -> Expected: X   | Output: X   | Correct?: True
Test 74  -> Expected: X   | Output: X   | Correct?: True
Test 75  -> Expected: X   | Output: X   | Correct?: True
Test 76  -> Expected: X   | Output: X   | Correct?: True
Test 77  -> Expected: E   | Output: E   | Correct?: True
Test 78  -> Expected: E   | Output: E   | Correct?: True
Test 79  -> Expected: E   | Output: E   | Correct?: True
Test 80  -> Expected: E   | Output: E   | Correct?: True
Test 81  -> Expected: B   | Output: D   | Correct?: False
Test 82  -> Expected: B   | Output: D   | Correct?: False
Test 83  -> Expected: B   | Output: I   | Correct?: False
Test 84  -> Expected: B   | Output: O   | Correct?: False
Test 85  -> Expected: K   | Output: K   | Correct?: True
```

Testergebnisse - Epoch 4 (Seite 3)

```
Test 86  -> Expected: K   | Output: R   | Correct?: False
Test 87  -> Expected: K   | Output: K   | Correct?: True
Test 88  -> Expected: K   | Output: X   | Correct?: False
Test 89  -> Expected: L   | Output: L   | Correct?: True
Test 90  -> Expected: L   | Output: L   | Correct?: True
Test 91  -> Expected: L   | Output: L   | Correct?: True
Test 92  -> Expected: L   | Output: L   | Correct?: True
Test 93  -> Expected: Y   | Output: Y   | Correct?: True
Test 94  -> Expected: Y   | Output: J   | Correct?: False
Test 95  -> Expected: Y   | Output: Y   | Correct?: True
Test 96  -> Expected: Y   | Output: P   | Correct?: False
Test 97  -> Expected: P   | Output: P   | Correct?: True
Test 98  -> Expected: P   | Output: P   | Correct?: True
Test 99  -> Expected: P   | Output: P   | Correct?: True
Test 100 -> Expected: P   | Output: P   | Correct?: True
Test 101 -> Expected: W   | Output: W   | Correct?: True
Test 102 -> Expected: W   | Output: W   | Correct?: True
Test 103 -> Expected: W   | Output: W   | Correct?: True
Test 104 -> Expected: W   | Output: W   | Correct?: True
results are 78.8 percent correct
```

H. Literaturverzeichnis

Adari, Y. (14. August 2023). *Beyond Backpropagation: Enhancing Neural Network Training with Optimizers [eigene Bearbeitung]*. Abgerufen am Juli 2025 von Medium: https://medium.com/@yaswanth.adari0/beyond-backpropagation-enhancing-neural-network-training-with-optimizers-2630500d475b

Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). *EMNIST: an extension of MNIST to handwritten letters*. Abgerufen am 2025 von http://arxiv.org/abs/1702.05373

Cretu, C. (6. April 2023). *How Does ChatGPT Actually Work? An ML Engineer Explains*. Abgerufen am Juni 2025 von Scalable Path Blog: https://www.scalablepath.com/machine-learning/chatgpt-architecture-explained

Donges, N. (15. August 2023). *4 Disadvantages of Neural Networks*. Abgerufen am Juni 2025 von Built In: https://builtin.com/data-science/disadvantages-neural-networks#:~:text=Neural%20Networks%20Require%20Lots%20of%20Data&text=Although%20there%20are%20some%20cases,would%20be%20the%20appropriate%20choice.

Everdu. (11. April 2025). *[기계학습심화] 2. 선형 회귀 & 경사하강법 [eigene Bearbeitung]*. Abgerufen am Juli 2025 von Everdu Blog: https://blog.everdu.com/574

Ian Goodfellow, Y. B. (2016). *Deep Learning.* Cambridge, MA: MIT Press. http://www.deeplearningbook.org.

IBM. (6. Oktober 2021). *Was sind neuronale Netzwerke?* Abgerufen am Juni 2025 von IBM Think: https://www.ibm.com/de-de/think/topics/neural-networks

IBM. (24. September 2024). *What is Claude AI?* Abgerufen am Juni 2025 von IBM Think: https://www.ibm.com/think/topics/claude-ai

Jung Min Han, Y. Q. (2021). Using recurrent neural networks for localized weather prediction with combined use of public airport data and on-site measurements. *Building and Environment, 192*, S. 107061. https://doi.org/10.1016/j.buildenv.2021.107601.

Kumar, S. K. (28. April 2017). *On weight initialization in deep neural networks*. Abgerufen am Juli 2025 von arXiv: https://arxiv.org/abs/1704.08863

Learning, L. (17. Juli 2022). *What is the Science of Reading? How the Human Brain Learns to Read*. Abgerufen am Juli 2025 von Lexia Learning Blog: https://www.lexialearning.com/blog/what-is-the-science-of-reading-how-the-human-brain-learns-to-read

Levine, D. (März 1989). Neural network principles for theoretical psychology. *Behavior Research Methods, Instruments, & Computers, 21*(2), S. 213–224. https://doi.org/10.3758/BF03205585.

MacNeil, C. (10. Januar 2025). *Wie die Sunk Cost Fallacy unsere Entscheidungen beeinflusst*. Abgerufen am September 2025 von Asana: https://asana.com/resources/sunk-cost-fallacy

Mazur, M. (17. März 2015). *A Step by Step Backpropagation Example*. Abgerufen am Juni 2025 von Matt Mazur Blog: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

NIST, N. I. (2. December 2024). *The EMNIST Dataset*. Abgerufen am Juni 2025 von NIST: https://www.nist.gov/itl/products-and-services/emnist-dataset

Raiders, W. (6. November 2023). *Letter Frequency in English*. Abgerufen am August 2025 von Word Raiders: https://wordraiders.com/guides/letter-frequency-in-english/

Sanderson, G. (3. November 2017). *Backpropagation calculus | Deep Learning Chapter 4*. Abgerufen am Juni 2025 von YouTube: https://www.youtube.com/watch?v=tIeHLnjs5U8

Sanderson, G. (n.d.). *3Blue1Brown*. Abgerufen am Juni 2025 von YouTube: https://www.youtube.com/@3blue1brown

Subba Reddy Oota, Z. C. (2023). *Deep Neural Networks and Brain Alignment: Brain Encoding and Decoding (Survey).* arXiv, Computer Science – Artificial Intelligence (cs.AI). arXiv.

University, S. (4. Juli 2025). *Feature Extraction Using Convolution [eigene Bearbeitung]*. Abgerufen am Juli 2025 von Stanford Deep Learning Tutorial: http://deeplearning.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution/

Unknown. (n.d.). *The Quotations Page*. Abgerufen am Oktober 2025 von Artificial Intelligence is no match for natural stupidity.: https://www.quotationspage.com/quote/239.html

Yagmur, U. (18. October 2022). *C, C++, Rust, Python, and Carbon (When to use Which?)*. Abgerufen am Juni 2025 von Medium: https://medium.com/codex/c-c-rust-python-and-carbon-when-to-use-which-2912a88f205b

Ich habe die Arbeit selbstständig und unter Aufsicht meines Betreuers verfasst und keine anderen als die angegebenen Hilfsmittel verwendet. Abschnitte, für deren Erstellung KI-Programme (bspw. ChatGPT) zum Einsatz kamen, habe ich allesamt offengelegt und mit einer entsprechenden Fussnote versehen. Meine Arbeit wird gegebenenfalls einer Prüfung bezüglich KI-Einsatz unterzogen; im Rahmen dieser Prüfung wird festgestellt, ob die Arbeit neben den angegebenen Stellen weitere von einer KI verfasste Elemente enthält. Ich nehme darüber hinaus zur Kenntnis, dass meine Arbeit zur Überprüfung der korrekten und vollständigen Angabe der Quellen mit Hilfe einer Software (eines Plagiaterkennungstools) geprüft wird. Zu meinem eigenen Schutz wird diese Software auch dazu verwendet, später eingereichte Arbeiten mit meiner Arbeit elektronisch zu vergleichen und damit Abschriften und eine Verletzung meines Urheberrechts zu verhindern. Ich erkläre mich damit einverstanden, dass die Schulleitung bei Verdacht auf Urheberrechtsverletzung meine Arbeit zu Prüfzwecken herausgibt.

Schlieren, 19.10.2025