HAW HAMBURG

BACHELORTHESIS
Lara Felina Schradick

Untersuchung der Auswirkung von Skalierung vor dem SoftMax Layer auf die Robustheit eines Neuronalen Netzes

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Lara Felina Schradick

Untersuchung der Auswirkung von Skalierung vor dem SoftMax Layer auf die Robustheit eines Neuronalen Netzes

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Technische Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Peer Stelldinger
Zweitgutachter: Prof. Dr. Bettina Buth

Eingereicht am: 18. Juli 2022

Lara Felina Schradick

Thema der Arbeit

Untersuchung der Auswirkung von Skalierung vor dem SoftMax Layer auf die Robustheit eines Neuronalen Netzes

Stichworte

Neuronale Netze, Robustheit, Maschinelles Lernen, SoftMax, Skalierungseffekt, Label Noise, Vanishing Gradients, Batch Normalisation

Kurzzusammenfassung

Vergleich von drei verschiedenen Ansätzen zur Stabilisierung neuronaler Netze gegen zwei unterschiedliche Arten von inkorrekt gelabelten Trainingsdaten. Dabei wollen alle drei Ansätze durch Abgrenzung von Logits ein besseres Trainingsergebnis erzielen. Zwei der Ansätze enthalten extra Schritte um Vanishing Gradients vorzubeugen.

Lara Felina Schradick

Title of Thesis

Investigating the effect of scaling before the SoftMax layer on the robustness of a neural network.

Keywords

Neural Networks, Stability, Maschine Learning, Softmax, scaling, label noise, vanishing Gradients, batch Normalisation

Abstract

Comparison of three different approaches to stabilize neural networks against two different types of incorrectly labeled training data. All three approaches aim to achieve a better training result by better differentiating logits. Two of the approaches include extra steps to prevent vanishing gradients.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar

Accuracy Prozentwert wie viele Testdaten korrekt klassifiziert worden.

Convolutional Neural Network Unterform von neuronalen Netzen, die mindestens eine Faltungsschicht beinhalten.

Logit Natürlicher Logarithmus einer Chance in der Statistik.

neuronales Netz Künstliche Nachbildung von Neuronen und Synapsen, in der Informatik.

Radizieren Wurzelziehen, Umkehrung der Potenzierung.

1 Einleitung

"Die Informationen der Welt verdoppeln sich etwa alle 20 Monate" - Thomas A. Runkler [16]

Im Jahr 2018 wurden 33 Zettabyte Daten digital generiert, für 2025 sind 175 Zettabyte generierte Daten prognostiziert [7]. Diese zunehmende Digitalisierung der Welt ermöglicht eine einhergehende stärkere Einbindung von künstlicher Intelligenz, da diese auf den generierten Daten trainiert werden kann. Doch bei der Menge ist eine Kontrolle auf Korrektheit nicht immer möglich, sodass davon ausgegangen werden kann, dass es fehlerhafte Daten im Trainingsprozess gibt.

Im Allgemeinen exisitieren mehrere Arten künstlicher Intelligenz. Der Fokus der vorliegenden Arbeit liegt auf den neuronalen Netzen; die dahinterliegende Motivation, Abschnitt 1.1, sowie die Ziele, Abschnitt 1.2, werden in diesem Kapitel beschrieben. Darüber hinaus wird der weitere Aufbau der Arbeit in Abschnitt 1.3 erläutert.

1.1 Motivation

Neuronale Netze sind eine Unterform von künstlicher Intelligenz, bei der versucht wird, Gehirnstrukturen nachzubauen. Gegenwärtig werden stetig mehr Aufgaben von neuronalen Netzen übernommen. So werden neuronale Netze zum Beispiel zur Vorauswahl von Bewerbenden genutzt, in der Handschriftenerkennung, im Bankwesen und besonders auch in der Medizin [15]. Dabei sind Vor- und Nachteile noch nicht abschließend diskutiert. So sind viele neuronale Netze in ihrer Entscheidungsfindung nicht nachvollziehbar oder verstärken menschliche Fehler und Vorurteile [21].
Denn während eine traditionelle, sequentielle Software eine Sequenz von wohldefinierten Anweisungen abarbeitet, besteht ein neuronales Netz aus einem gerichteten Graphen,

welcher Berechnungen in Abhängigkeit von der Gewichtung, ermittelt durch das Training, der einzelnen Kanten ausführt [8].
Das Verhalten eines neuronalen Netzes entsteht dementsprechend durch die Topologie und der verwendeten Trainingsdaten, somit hängt die Qualität maßgeblich von diesen beiden Faktoren ab [4].
Zum Trainieren eines neuronalen Netzes wird eine große Menge an Trainingsdaten benötigt. Diese können verschiedene Arten von Fehlern erhalten, deren Korrektur viel Zeit und Aufwand benötigt [14].

Dabei stellt sich die Frage, ob Probleme in der Qualität der Trainingsdaten durch eine veränderte Topologie ausgeglichen werden können, was neuronale Netze nicht nur stabiler, sondern auch kostengünstiger und zeiteffizienter im Training machen würde. Genau diese Thematik bildet den Anfangspunkt und die Motivation für die Durchführung dieser Arbeit.

1.2 Ziel der Arbeit

Der oben in Abschnitt 1.1 erwähnte Ansatz zur Stabilisierung des neuronalen Netzes durch Veränderung der Topologie wurde bereits durch verschiedene wissenschaftliche Arbeiten theoretisiert und untersucht. Dabei wird sich jedoch zumeist auf einen einzigen Ansatz zur Veränderung der Topologie konzentriert. Eine Gegenüberstellung verschiedener Ansätze könnte weitere Ideen und Aufschlüsse über den Einfluss der Topologie auf die Qualität bringen.

Das grundlegende Ziel dieser Arbeit ist es daher, drei Ansätze (s. unten), die alle eine Skalierung beinhalten, miteinander zu vergleichen und im Hinblick auf ihre Auswirkung auf die Robustheit des neuronalen Netzes sowie den Umgang mit verschiedenen Mengen an Label Noise auszuwerten. Insbesondere soll überprüft werden, ob durch den Skalierungseffekt die Logits unterschiedlicher Klassen stärker voneinander abgegrenzt werden, was zu einem stabileren Netz führen kann [17].

Skalierung und Batch Normalization Der erste Ansatz wurde 2020 in einem Paper vorgestellt und hat gezeigt, dass bei der Skalierung vor einer Batch-Normalization-Schicht

im neuronalen Netz die Klassifikationen des Netzes verbessert werden können [20]. In diesem Paper wird jedoch, anders als in dieser Arbeit, nicht auf Stabilität eingegangen, sodass noch keine Werte vorliegen.

Skalierung vor Softmax-Funktion mit anschließender Wurzelziehung Der zweite Ansatz nutzt die oft bereits im neuronalen Netz verwendete Softmax Funktion. Durch die Skalierung dieser wird versucht, die Werte so zu beeinflussen, dass das Netz sicherer in seiner Klassifikation wird. Um die Auswirkungen der Skalierung auch in weiteren Durchläufen sinnvoll nutzen zu können, wird anschließend radiziert.

Skalierung vor Softmax Funktion Der dritte Ansatz dient schließlich der Überprüfung des zweiten Ansatzes, die Radizierung wird hierbei unterlassen.

1.3 Struktur der Arbeit

Im Folgenden werden der Aufbau der Experimente mit Begründung und die aus den Grundexperimenten folgenden weiteren Experimente sowie deren Beobachtung aufgeführt. Diese Beobachtungen werden dann genauer diskutiert und eingeordnet.

Im folgenden **Kapitel 2** soll gewecktes Interesse mit Grundwissen zum Thema und der Herangehensweise verstärkt und ausgebaut werden.
Aufbauend auf das dadurch entstandene Grundverständnis und Hintergrundwissen wird in **Kapitel 3** als Vorbetrachtung auf das Experiment eine Analyse der Randbedingungen durchgeführt und mit Hinblick auf den in **Kapitel 4** folgenden Versuchsaufbau einzelne Bereiche vertieft betrachtet. Dort wird der Versuchsaufbau vorgestellt, anhand dessen Versuche durchgeführt und Beobachtungen zu diesen notiert werden.
In **Kapitel 5** werden die Beobachtungen aus dem vorherigen Kapitel interpretiert und mit den Annahmen aus Kapitel 3 verglichen. Die Arbeit wird mit bereits bekannten Werten und Theorien verglichen. Zudem werden Fehler aufgezeigt und mögliche Verbesserungen vorgeschlagen.
Im letzten Kapitel, **Kapitel 6**, wird durch sowohl subjektive, als auch bemüht objektive Diskussion der Ergebnisse ein Schlussstrich gezogen. Zudem werden durch die Arbeit neu gewonnene Inhalte und Fragestellungen strukturiert zusammengefasst.

2 Grundlagen

In diesem Kapitel werden die notwendigen Fachbegriffe zu den zentralen Themen der Arbeit definiert.
Als erstes wird der Begriff "Neuronales Netz" mit dem für diese Arbeit genutzten Convolutional Neural Networks geklärt. Daran anschließend wird auf den Aufbau einzelner, benötigter Layer sowie den Begriff Robustheit eingegangen.

2.1 Neuronales Netz

Ein neuronales Netz ist eine Ansammlung an miteinander verbundenen Neuronen. Wird durch die Information an den Eingängen ein Schwellenwert überschritten, feuert ein Neuron und gibt Information an ein verbundenes Neuron weiter.
Diese sowohl bei Menschen als auch bei Tieren vorkommende Vernetzung bilden Nervensysteme. Abgeleitet aus dem biologischen Beispiel wird in der Informatik ein künstliches neuronales Netz erschaffen, welches aus miteinander durch Kanten verknüpften Neuronen besteht. Die Gewichtung der Kanten zueinander kann sich unabhängig der Vorgaben im Verlauf eines Trainings ändern. Dieser Prozess wird als Lernen bezeichnet. [8]

Das Besondere an neuronalen Netzen ist, seien sie natürlich oder künstlich, ihre Fähigkeit, Muster zu erkennen ohne die dem Muster zugrunde liegenden Regeln zu kennen [8].

Layer Neuronale Netze sind in der Regel in Schichten, sogenannten Layern, aufgebaut. Dabei umfasst ein Layer mehrere Neuronen, welche über Synapsen mit den Neuronen der nächsten Schicht verbunden sind.

Die verschiedenen Layer haben unterschiedliche Aufgaben innerhalb des neuronalen Netzes. Dabei sind die Eingabewerte eines Layers, verändert um die Gewichtung der Verbindungskante, die Ausgabewerte des vorherigen Layers.

2.1.1 Convolutional Neural Network

Eine spezielle Form von neuronalen Netzen ist das gefaltete neuronale Netz, im Folgenden in der gebräuchlicheren, englischen Form, Convolutional Neural Network benannt. Meistens wird diese Art eines neuronalen Netzes, inspiriert vom tierischen und menschlichen Sehvermögen, zur Bilderkennung oder Verarbeitung von Bild- und Videodaten genutzt. [8]

Um als Convolutional Neural Network zu gelten, muss eine Netztopologie mindestens eine Faltungsschicht besitzen. Eine Faltungsfunktion wird dabei als Filter genutzt, welcher unter Verwendung einer Gewichtsfunktion über den gesamten Inhalt multiplilziert.
Anstelle alle Kantengewichtungen einer Matritzenmultiplikation speichern und immer wieder überarbeiten zu müssen, müssen nur die Kantengewichtungen einer deutlich kleineren Funktion, dem Kernel, gespeichert werden, welcher über alle Pixel angewendet wird.

Convolution Eine Convolution, also Faltungsfunktion, besteht aus einer Eingabe, die sowohl ein- als auch mehrdimensional sein kann, einer Funktion über diese Eingabe mit unterschiedlichen Gewichtungen, dem Kernel und der Ausgabe.

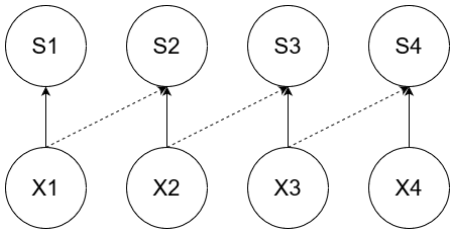

Abbildung 2.1: Vereinfachte Darstellung eines 2 Neuronen breiten Kernels

Ohne die in anderen Netzen verwendete Matritzenmultiplikation über die komplette Eingabe, spart ein Convolution Layer nicht nur Speicherplatz und Zeit, sondern kann auch mit Eingaben wechselnder Größe im gleichen Netz trainieren.
Goodfellow et. al.[8] beschreibt diesen und weitere Gründe, weswegen Convolutional Neural Networks als hilfreich angesehen werden:

- **Sparse Interaction** Die für Convolutional Neural Network relevante Verwendung eines Kernels, welcher kleiner ist als der Input, wird als sparse interaction oder sparse connectivity bezeichnet, da nicht mehr jede Eingabe mit jeder Ausgabe verknüpft ist, wie bei einer Matritzenmultiplikation. Durch die geringere Anzahl an Operationen, die benötigt werden, um die Ausgabe des Layers zu generieren, sorgt sparse interaction für eine hohe Effizienz, sowohl bei genutztem Speicherplatz, als auch in der Laufzeit.
- **Parameter Sharing** Statt einzelne Gewichtungen für jede Eingabeposition zu lernen, werden die Gewichte zwischen den Eingabewerten wiederholt. Bei gleichbleibender Trainingsdauer wird die Menge an Speicherplatz um ein Vielfaches reduziert.
- **Equivariance to Translation** Als Folge des Parameter Sharings sind Convolutional Layer äquivariant, also aufeinander abbildbar. Verändert sich die Eingabe, verändert sich die Ausgabe im gleichen Maße. Wird zum Beispiel auf einem Bild ein Bereich verschoben, so ist diese Verschiebung in der Ausgabe wiederzufinden. Wird das Bild jedoch gedreht oder neu skaliert, kann das Convolutional Layer dies nicht mehr equivariant abbilden.

Pooling Neben dem Convolutional Layer und einer Aktivierung durch eine nichtlineare Aktivierungsfunktion haben viele Convolutional Neural Networks zudem ein Pooling Layer. Dieses ersetzt die Ausgabe an einem speziellen Punkt des Netzes durch die Ausgabe einer statistsichen Funktion über die benachbarten Ausgaben. Das Layer bringt die Ausgabe so zusammen. Häufig genutzte Funktionen sind das Maximum, abgebildet in Abbildung 2.2, oder der Durchschnitt eines rechteckigen Bereiches. [8]

Durch Pooling wird die Ausgabe invariant zu kleinen Veränderungen der Eingabe. Aufgrund dessen ist die Existenz eines Features relevanter als der exakte Punkt, an dem das Feature erkannt wird. [8]

Um die Effizienz des Netzes weiter zu erhöhen, können in Pooling Layern die Ausgabeneuronen reduziert werden, indem die statistische Funktion, in Abbildung 2.2 als verschiedene Rechtecke dargestellt, nicht überschneidend auf Neuronen angewendet wird, sodass mehrere Eingabeneuronen auf ein einziges Ausgabeneuron zusammengefasst werden. Dadurch

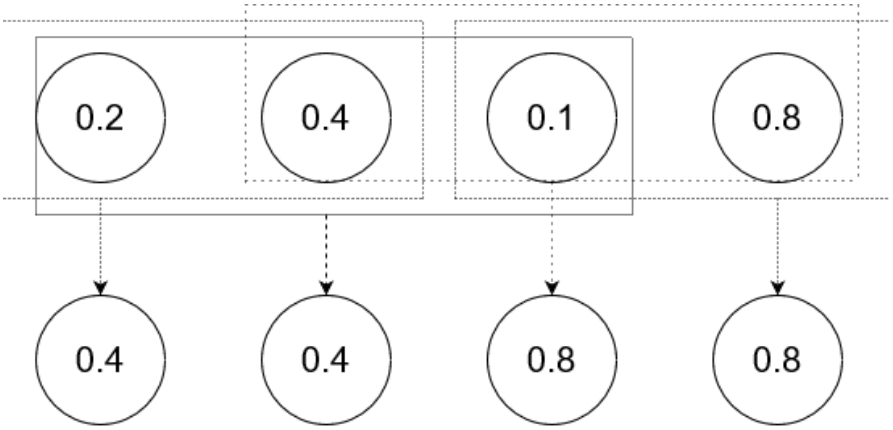

Abbildung 2.2: Vereinfachte Darstellung eines Maximum Pooling über 3 Neuronen

können unterschiedlich große Eingabedaten auf eine identische Größe minimiert werden. [8]

Rückpropagierung Im Gegensatz zu mach anderen neuronalen Netzen funktioniert Rückpropagierung für Convolutional Neural Networks. Dadurch können Fehler effektiver beim Lernen genutzt werden.
Zuerst wird das Eingabemuster durch das Netz gegeben, die Ausgabe des Netzes, also dessen berechnete Wahrscheinlichkeiten, werden mit der gewünschten Ausgabe verglichen. Die Differenz bildet den Fehler des Netzes. Aus diesem Fehler wird auch der sogennante Loss berechnet, anhand dessen überprüft wird, wie gut ein Netz lernt.
Dieser Fehler wird durch das Netz zurückgereicht, er wird rückpropagiert. Bei dieser Rückpropagierung werden die Kantengewichtungen, je nach Einfluss auf den Fehler, verändert. Die Funktion mit der dieser Einfluss berechnet wird, ist die Optimierung. [8]
Bei einem erneuten Training des Netzes sollte dieses nun bessere Werte erzielen können, da die Kantengewichtungen aus dem letzten Fehler gelernt haben.

Insgesamt lässt sich festhalten, dass das Convolutional Layer und die Rückpropagierung ausreichend sind, um alle Gewichtungen zu berechnen, die zum Trainings eines Convolutional Neural Network gebraucht werden. Durch ein Pooling Layer kann das Training noch effizienter durchgeführt werden [8].

2.1.2 Zu erwartende Probleme

Bei Convolutional Neural Network und ihrem Training können verschiedenste Probleme auftreten. Für diese Arbeit sind drei dieser Probleme besonders relevant.

Overfitting und Underfitting Wird ein Netz zu häufig oder mit zu spezifischen Trainingsdaten trainiert, so kann es dies dazu führen, dass eine einfachere Form nicht mehr erkannt wird. Dieses Problem, die Überspezialisierung eines Netzes, wird als Overfitting bezeichnet [8]. Exakt konträr zum Overfitting, verhält es sich mit neuronalen Netzen, die underfittet sind. Ihnen fehlt das Training, um genug gelernt zu haben, daher ist ihr Fehler noch zu groß, um sinnvolle Ergebnisse zu erzielen [8].

Vanishing Gradient Effekt Das Problem der Vanishing Gradients entsteht, wenn bei der Rückpropagierung während des Trainings eines neuronalen Netzes die Werte zur Kantenveränderung zu klein sind. Dadurch stagniert der Lernprozess des neuronalen Netzes [3].

2.1.3 Softmax

Als Möglichkeit, die Ausgabe eines neuronalen Netzes in Wahrscheinlichkeit umzuwandeln, findet häufig die Softmax Funktion in neuronalen Netzen Anwendung, die zur Klassifizierung genutzt werden. Dabei bildet die Funktion die Wahrscheinlichkeitsverteilung der Werte K eines neuronalen Netzes ab, indem ein Vektor von K reellen Zahlen, den sogenannten Logits, in einen Vektor von K positiven reellen Zahlen mit der Summe 1 zugeordnet wird. Diese werden dann als Wahrscheinlichkeiten interpretiert. Für diese Zuordnung wird eine Exponentialfunktion verwendet [18].
So kann die Softmax-Funktion aus den beliebigen Ausgaben eines vorherigen Layers einen Wahrscheinlichkeitsvektor erzeugen, bei dem jeder möglichen Klassifikation eine Wahrscheinlichkeit zwischen 0 und 1 zugeordnet ist.

Wenn die Eingabewerte in die Sofmax Funktion betragsmäßg groß sind, kann ein Skalierungseffekt betrachtet werden, dieser wird auch als Sättigung bezeichnet [18]. Betragsmäßig große Zahlen werden größer zugeordnet als aus dem Eingabeverktor erkennbar,

während betragsmäßig kleine Zahlen dichter beieinander liegen.

2.1.4 Batch Normalization

Da über den Verlauf der Layer die Daten kontinuierlich verändert werden, ist die Distribution nicht mehr ausgeglichen, sondern häufig punktuell konzentriert. Dieser sogenannte internal covariate shift erschwert das Trainieren von Neuronales Netz.
Das Ziel der Batch Normalization ist es, durch die Normalisierung der Eingabe eines Layers das Netz schneller und stabiler trainieren zu lassen. Dazu wird die Eingabe skaliert und neu zentriert [19].

Die Normalisierung der Daten vor Beginn des Tests oder des Trainings ist üblich, um besondere Features zu betonen und Differenzen von Eingaben zu minimieren. Bei der Batch Normalization wird diese Normalisierung anstelle auf die Eingabedaten auf Werte innerhalb eines Layers und, anstelle über alle Daten zu einem Zeitpunkt, lediglich über ein Batch zur Zeit, also eine kleine Menge an Daten, durchgeführt. Dadurch erleben, neben normalisierten Werten, besonders tiefere Layer eine minmalere Differenz der Eingabedaten sowie eine Regularisierung, die helfen kann, Overfitting zu verhindern [19].

2.1.5 Label

Für diese Arbeit wird zwischen Feature Labels, die spezifischere Inhalte innerhalb von Trainingsdaten beschreiben, und Class Labels, welche die Traininsdaten in verschiedene Klassen einteilen, unterschieden. So gibt ein Feature Label zum Beispiel den Farbwert eines einzelnen Pixels oder Pixelbereiches an, während ein Class Label angibt, das die Gesamtheit der Pixel eine 6 darstellt.

Label Noise

Als Label Noise werden entweder alles, was die Beziehung zwischen Features und einer Klasse verzerrt, oder nicht systematische Fehler bezeichnet [12]. Sowohl Feature als auch Class Labels können Noise enthalten. Dabei ist der allgemeine Konsens, dass Feature

Noise zu einem gewissen Grad dabei hilft, Overfitting zu verhindern, Class Label Noise jedoch das korrekte Lernen verhindert [4].

Nach Zhu et.al.[20] wird zwischen vier Gründen von Label Noise unterschieden:

- menschliche Fehler (falsche Klassifizierung)
- inkohärentes oder unzureichendes Wissen über den Inhalt und dementsprechend unmögliches korrektes Klassifzieren
- Möglichkeit, eine Information mehreren Klassen zuzuordnen
- Kommunikations- oder Verschlüsselungsfehler

Im Rahmen dieser Arbeit wird zum einen mit randomisiertem Noise auf Class Label Ebene gearbeitet. Dafür wird eine definierte Menge an Class Labels zufällig durch andere mögliche Class-Labels ersetzt, ohne die tatsächlichen Arten von Label Noise, die entstehen, zu beachten. Zum anderen wird auf Label Noise, welches aufgrund menschlicher Fehler beim Klassifizieren entstehen könnte, zurückgegriffen. In Kapitel 4 wird auf die Umsetzung der Label Noise-Generierung genauer eingegangen.

2.2 Robustheit

Sowohl der Duden [1], als auch Oxford Languages [2], definieren robust als *"kräftig, stabil; nicht empfindlich oder leicht irritierbar"*.
Für diese Arbeit lässt sich Robustheit als ein Convolutional Neural Network ableiten, welches Eingabedaten korrekt klassifiziert, unabhängig von der falschen Modifizierung der Daten. Es handelt sich also um ein Netz, das möglichst lange möglichst viele Daten korrekt klassifiziert, auch wenn die Daten, auf denen es trainiert, immer inkorrekter werden, da sie zunehmend mehr Label Noise beinhalten.

Dabei ist Robustheit ausschließlich auf Stabilität gegenüber Veränderung von Class Labels bezogen. Ein Verändern der Bilder durch Verschieben, Skalieren oder Ähnliches ist nicht vorgesehen.

3 Vorbetrachtung

Aufbauend auf den vorangehenden Grundlagen werden im Folgenden die genutzen neuronalen Netze, deren Parameter sowie Ansätze analysiert. Ebenfalls werden im Unterabschnitt 3.1.7 grundlegende Thesen über die zu erwartenden Ergebnisse aufgestellt und in den einzelnen Abschnitten der drei Ansätzen erweitert.

3.1 Analyse

Zuerst werden die beiden Basisnetze präsentiert, die den Experimenten zugrunde liegen. Daran anschließend werden sowohl die verwendete Parameter als auch die Layer, die verändert oder hinzugefügt werden, vorgestellt, um die drei Ansätze zur Verbesserung der Robustheit zu vergleichen.

3.1.1 Neuronale Netze

Es werden zwei Basisnetze implementiert und entsprechend der folgenden Formeln in ihren Layern ergänzt.
Bei beiden Basisnetzen handelt es sich, entsprechend der in Unterabschnitt 2.1.1 vorgestellten Topologie von Goodfellow [8], um Convolutional Neural Networks. Durch die geringe Größe beider Basisnetze ist ein schnelles Training mit ausreichender Performance möglich.

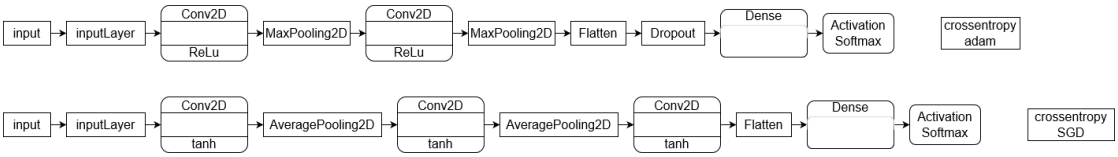

Abbildung 3.1: Einfache Darstellung beider Basisnetze in ihrer Grundstruktur

Bei dem in Abbildung 3.1 zu sehenden oberen Netz handelt es sich um das gemeinhin als LeNET bekannte CNN [10], welches auch in anderen Arbeiten und in der Forschung als Basisnetz genutzt wird. Das andere Netz entspricht im Aufbau der Anleitung, die auf der Keras eigenen Website zu finden ist [5], und wird dort zu Erklärung des unten beschriebenen Datensatzes MNIST genutzt. Dieses zweite Netz wird weiterführend zum einfacheren Identifizieren als Keras-Basisnetz bezeichnet.

Convolutional Die Convulutional Layer jedes Netzes falten die Eingabe, wie in Absatz 2.1.1 erklärt, das 2D beschreibt dabei die Art der Eingabe. Da es sich um ein Bild handelt, ist die Eingabe zweidimensional. ReLu und tanh sind, ähnlich der bereits beschriebenen Softmax-Funktion, Aktivierungsfunktionen. In dieser Abbildung steht tanh für Hyperbolic tangent und weist der Ausgabe einen passenden Wert zwischen -1 und 1 zu. Die im Keras-Netz verwendete Aktivierungsfunktion Rectified Linear Unit, kurz ReLu, dagegen weist Werte zwischen 0 und 1 zu und benötigt weniger Rechenaufwand als tanh, dafür bildet sie Werte kleiner 0 nicht oder inkorrekt ab. [8]

Pooling Wie in Absatz 2.1.1 beschrieben, verwendet dieses Layer eine statistische Funktion, um die Ausgaben zusammenzuführen. Dabei wird im Keras-Netz immer das Maximum benachbarter Neuronen als Ausgabe verwendet, während für das LeNet der Mittelwert der benachbarten Neuronen berechnet und als Ausgabe verwendet wird. Das 2D bezeichnet auch hier den Aufbau der Eingabe, die weiterhin zweidimensional ist. [8]

Flatten Das Layer Flatten reduziert den Input um eine Dimension, ohne dabei die Batch Size zu verändern. Das bisher zweidimensionale Bild wird in diesem Falle zu einer Dimension geglättet. [6]

Dropout Dieses Layer setzt zufällig ausgewählte Eingabewerte auf 0, um Overfitting vorzubeugen. Um die Gesamtsumme der Inputs nicht zu verändern, werden die weiteren Eingabewerte hochskaliert. [6]
Das Dropout Layer wird ausschließlich vom Keras-Basisnetz genutzt.

Dense Im Dense Layer, auch fully connected Layer, sind dem englischen Namen entsprechend alle Neuronen mit allen vorherigen Ausgangsneuronen verknüpft. Das Dense Layer führt eine Matrixmultiplikation aus. Dadurch wird die Form der Eingabe verändert. [6]

Kompilierung des Models Das Modell des Netzes wird anschließend kompiliert, ein Schritt, der in Abb. 3.1 als freistehende Box gekennzeichnet ist, und in Abhängigkeit von Optimierungs- und Lossfunktionen trainiert.

Cross Entropy Loss Zur Berechnung des Loss wird Cross Entropy Loss bei beiden Basisnetzen genutzt.
Dabei ist Entropie eine Angabe von Unordnung oder Unvorhersehbarkeit eines Systems. Desto größer die Entropie, desto unordentlicher ein System. Dabei kann die Entropie ohne externe Einwirkung nicht minimiert werden, sondern ausschließlich steigen, ein System kann also nur unordentlicher werden [6].

Cross-Entropy Loss wird auch logarithmischer Loss genannt und vergleicht für jede Klasse die vom Netz gegebene Wahrscheinlichkeit mit den korrekten Werten von 0 oder 1. Die Differenz dieser beiden Werte bilden den Grundwert, aus dem die Bestrafung des Netzes berechnet wird. Dabei ist dieser bestrafende Wertlogarithmus größer, wenn die Differenz ebenfalls dichter an 1 ist und tendiert zu 0 bei kleinen Differenzen.
Es wird für jedes Klassenlabel einzelnd der Loss berechnet und anschließend als Gesamtloss addiert [6].

Optimierung In Abhängigkeit der Optimierungsfunktion werden bei jeder neuen Eingabe eines Batches in das neuronale Netz die Gewichtungen der Kanten zwischen den Neuronen verändert. Diese Veränderung stellt das umgangssprachliche Training dar.

Der *SGD*, Stochastic Gradient Descent, selektiert zufällig einen Datensatz aus mehreren und probiert verschiedene Gewichtungen aus, um die beste auszuwählen. Dabei ist möglichst geringer Loss der entscheidende Faktor
Die Optimierung *Adam* betrachtet bei der Setzung neuer Gewichte vorheriges und aktuelles Loss. Dabei nutzt es den regularisierten Mittelwert. Adam kann sehr exakt von außen eingestellt werden. [8]

3.1.2 MNIST

Die Modified National Institute of Standards and Technology database, kurz MNIST Datenbank, enthält 70.000 handgeschriebene Ziffern in 28*28px großen Fotos in Graustufen. Die Bildmenge teilt sich auf 60.000 Trainings- und 10.000 Testbilder auf, die der NIST Datenbank entnommen worden. Alle Ziffern sind von Nordamerikaner*innen geschrieben worden und entsprechen der dort üblichen Ziffernschreibweise von arabischen Ziffern. Der für Forschungszwecke eher kleine Datensatz wird häufig zur Überprüfung von Thesen oder kategorischen Convolutional Neural Network verwendet. [11]

3.1.3 Epochenanzahl und Batchgröße

Epochen definieren die Häufigkeit, mit der ein neuronales Netz trainiert wird, also wie häufig es alle 60.000 Trainingsbilder durchläuft.
Die Größe eines Batch gibt an, wie viele Bilder parallel das neuronale Netz passieren.
Daraus ergibt sich eine initiierte Trainingsgesamtmenge von
$(trainingssize/batchgre) * epochen$ [8],
die bisher einzig bekannte Größe die Trainingssize in Abhängigkeit der Trainingssize von MNIST ist.

Je kleiner ein Batch ist, desto intensiver wird das neuronale Netz trainiert, das Training nimmt infolgedessen aber mehr Zeit in Anspruch. Gegenproportional verhält es sich mit den Epochen. Je weniger Epochen, desto weniger Zeit benötigt das Training. Dafür wird das neuronale Netz auch weniger häufig trainiert.

Um in dieser Arbeit eine Vergleichbarkeit zwischen den verschieden trainierten neuronalen Netzen zu gewährleisten, sollen alle neuronalen Netze mit identischer Batchgröße und Epochenanzahl trainiert werden. Zur Bestimmung passabler Werte wurde das Basisnetz LeNet 3.1.1 mit variablen Größen trainiert und nach jedem Epochendurchlauf auf Testdaten validiert.

Anhand von Abb. 3.2 ist dabei zu erkennen, dass alle drei Batchgrößen einen Sättigungseffekt erfahren, durch welchen sie sich in höheren Epochen substantiell weniger verbessern als zu Beginn des Trainings. Dieser Effekt tritt stärker bei kleineren Batchgrößen auf, welche wie erwartet dennoch bessere Ergebnisse erzielen.

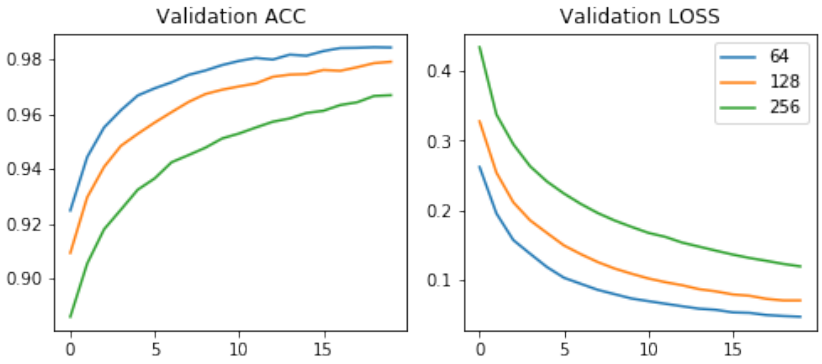

Abbildung 3.2: Loss und Validation in Prozent, bei 1-20 Epcohen und einer Batchgröße von 64, 128 oder 156

Aufgrund des Einsetzen des Sättigungseffektes nach acht bis zehn Epochen werden folgende neuronale Netze für zehn Epochen trainiert.

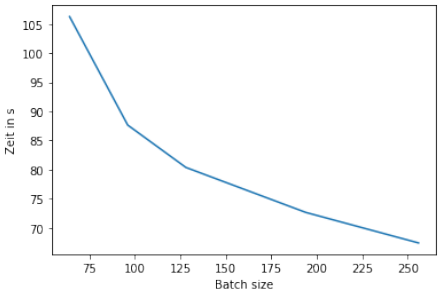

Abbildung 3.3: Trainingsdauer von 10 Epoche in Abhängigkeit der Batchgröße

Da 64 und 128 Bilder als Batchgröße einen Unterschied von weniger als 2 Prozentpunkte zu jeglichem abgebildeten Zeitpunkt ausmachen, wie in Abb. 3.2 zu sehen, und die Trainingsdauer von kleineren Batchsizes rasant steigt, scheint der um 50% größere Zeitaufwand die leichte Verbesserung nicht zu rechtfertigen. Dies ist insbesondere der Fall, wenn berücksichtigt wird, dass Netze häufiger mit verschiedenen Parametern trainiert werden und sich der Zeitaufwand so addiert.

Aus diesen Betrachtungen erschließt sich also, dass für die folgenden Trainingsdurchläufe eine Epochenanzahl von zehn und eine Batchgröße von 128 genutzt werden.

3.1.4 Scaled Batch Norm

In ihrem Paper stellen Zhu et. al.[20] einen Ansatz zur Verbesserung der Klassifikation eines Convolutional Neural Networks vor. Ihre Beobachtung ist, dass eine Skalierung vor dem Softmax-Layer zu einer verbesserten Klassifikation führen könnte, wenn durch diese Skalierung die Vanishing Gradients nicht ein Problem wären.
Um das Problem der Vanishing Gradients zu umgehen, schlagen Zhu et. al.[20] eine skalierte Batch Normalization vor, im Folgenden Scaled Batch Norm genannt.

In ihren Versuchen ist es gelungen, durch die Scaled Batch Norm eine Verbesserung der Klassifikation zu erzielen. Anders als in dieser Arbeit, wurde dabei auf einem CIFAR100 Datensatz gearbeitet, welcher ähnlich zu MNIST pixelbasierte Bilder enthält, jedoch einen größeren Umfang hat, coloriert ist und verschiedene Tiere und Gegenstände beinhaltet [9]. Außerdem wurden die Netze mit bis zu 100 Epochen und abweichenden Parametern trainiert.
Dennoch ist davon auszugehen, dass Scaled Batch Norm eine Verbesserung der Klassifizierung zu den beiden Basisnetzen zeigt. Da im ursprünglichen Paper nicht auf Robustheit eingegangen wird, liegen darüber noch keine Daten vor. Jedoch führt eine bessere Grundklassifizierung zu höheren Ausgangswerten. Demnach lässt sich annehmen, dass, solange das Label Noise noch nicht überhand nimmt, weiterhin eine Verbesserung erkennbar ist.

Dem Paper von Zhu et. al.[20], entnommen, wird die Batch Normalization, dargestellt in Abbildung 3.4, als vorletztes Layer, noch vor der SoftMax-Aktivierungsfunktion, eingefügt.

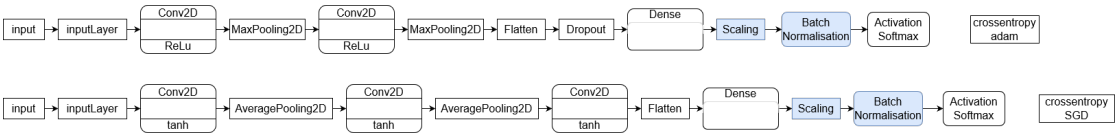

Abbildung 3.4: Einfache Darstellung beider Basisnetze, in Blau die Veränderung durch Scaled batch Norm

3.1.5 Skalierung mit anschließendem Wurzelziehen

Ein weiterer Ansatz, vorgestellt von Stelldinger[18], ist eine reine Skalierung vor dem SoftMax-Layer ohne Zuhilfenahme der Batch Normalization. Um dem Problem der Va-

nishing Gradients zu entgehen, wird zusätzlich mit der Wurzel des Skalierungsvektors multipliziert, sodass die Werte des Loss ausreichend groß sind.[18]

Zur Umsetzung werden zwei Veränderungen, in blau in Abbildung 3.5, vorgenommen. Die Skalierung erfolgt mit der Division durch den Skalierungswert und nach der Softmax-Funktion wird mit der Wurzel des Skalierungswertes s multipliziert.

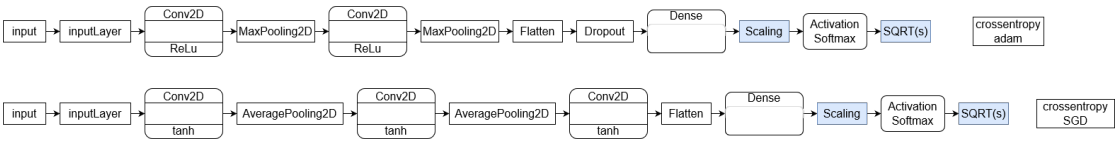

Abbildung 3.5: Einfache Darstellung beider Basisnetze, in Blau die Veränderung dieses Ansatzes

3.1.6 Reine Skalierung

Da beide Ansätze sich darauf beziehen, dass eine Skalierung der Softmax-Funktion zur Verbesserung der Klassifizierung beiträgt und beide Ansätze des Weiteren versuchen, Vanishing Gradients zu verhindern, wird als dritter Ansatz eine Skalierung vor der Softmax-Funktion ohne Ausgleich für Vanishing Gradients evaluiert.

Da Vanishing Gradients ein großes Problem der Rückpropagierung darstellen, ist davon auszugehen, dass dieser Ansatz keine bessere Klassifizierung erreicht als das Basisnetz, unabhängig des Label Noise.

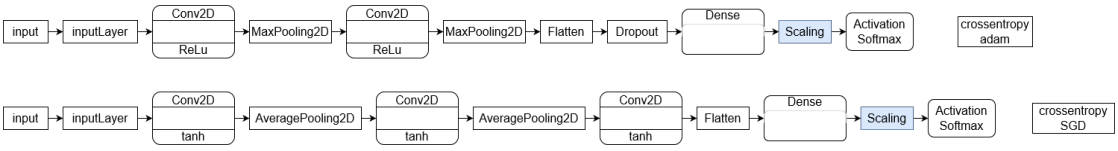

Abbildung 3.6: Einfache Darstellung beider Basisnetze, in Blau die Einfügung der Skalierung

Um weiterhin möglichst Dicht am vorherigen Ansatz zu bleiben, wird auch in diesem nicht mit dem Skalierungsvektor multipliziert, sondern dividiert.

3.1.7 Erwartungen an die unterschiedlichen neuronalen Netze

Trotz dessen, dass es sich bei beiden Basisnetzen um Convolutional Neural Networks handelt, gibt es auch deutliche Unterschiede. Das Keras-Netz wird sowohl durch die Verwendung der ReLu Aktivierungsfunktion, als auch die geringe Menge an Faltungsschichten effektiver arbeiten als das LeNet.
Besonders für den dritten Ansatz, welcher keine eigenen Schritte zur Beseitigung des Vanishing Gradients unternimmt, könnte die Verwendung von ReLu im Keras nützlich sein, da ReLu, anders als tanh, dazu beiträgt, das Problem der Vanishing Gradients zu unterdrücken. Dadurch ist davon auszugehen, das dieser Ansatz mit Keras besser und länger lernt als mit LeNet.

Durch die Verwendung von Dropout bleibt das Keras-Netz eventuell länger stabil gegen die Einführung von Label-Noise. Da die Batch Normalization des ersten Ansatzes aber einen ähnlichen Effekt erreicht [8], könnte LeNet und dieser Ansatz nahezu identische Ergebnisse erzielen.

3.2 Messung der Robustheit

Auf Basis der für diese Arbeit in Abschnitt 2.2 gegebene Definition von Robustheit eines Convolutional Neural Networks können in Kapitel 4 trainierte Convolutional Neural Networks anhand ihrer Accuracy- und Losswerte auf Robustheit verglichen werden.
Da durch die Eingabe von Label Noise der Loss bewusst manipuliert wird, sind ausschließlich die Accuracy-Werte, also wie viele Bilder das Convolutional Neural Network richtig klassifiziert, für die Bewertung der Robustheit relevant.
Zudem werden alle Netze unter den gleichen Bedingungen trainiert, sodass ein direkter Vergleich innerhalb einer Art von Basisnetz (LeNet oder Keras) möglich ist. Bei einem Vergleich über die Grenzen der Basisnetze hinaus müssen die grundliegenden Unterschiede der Accuracy berücksicht werden, sollte ein Basisnetzer dauerhaft schlechtere Ergebnisse abliefern.

Darüber hinaus werden besonders interessante Skalierungen häufiger trainiert, um eine Standartabweichung festzustellen. Diese kann angeben, wie stabil das Netz mit dieser Skalierung in der Lage ist, konstante Klassifikationsleistungen zu erbringen.

4 Aufbau und Durchführung

In **Kapitel 4** wird, mit Bezug auf die in Kapitel 3 erfolgte Analyse, der Versuchsaufbau der durchgeführten Experimente konkret begündet und vorgestellt. Zudem werden die Beobachtungen bei den verschiedenen neuronalen Netzen festgehalten, jedoch noch nicht interpretiert.

4.1 Aufbau der Experimente

Der Aufbau der einzelnen Experimente wird begrenzt durch die in Unterabschnitt 3.1.3 bestimmte Anzahl an Epochen und Batchgröße, sowie der Tatsache, dass aufgrund der Radizierung im zweiten Ansatz keine negative Skalierung erfolgen kann.
Aus dem Paper zum ersten Ansatz, der Scaled Batch Norm, lässt sich erschließen, dass die besten Werte im kleineren Zahlenraum erzielt wurden. Das ist grob auch für scaled Softmax anzunehmen. Um sicherzugehen, dass alles Relevante betrachtet werden kann, werden Skalierungen zwischen 0.1 und 10 vorgenommen.
Um die Robustheit einschätzen zu können, den Zeitaufwand dabei aber möglichst gering zu halten, werden die Label Noise bei Experimenten mit Skalierungen in 10 Prozentpunktschritten erhöht. Die Basisnetze als Ausgangswerte werden genauer trainiert, da bei ihnen keine Skalierung erfolgt.

Zusammengefasst wird ein Ansatz auf einem Basisnetz also zehn Epochen trainiert, dann neu skaliert, ein neues Netz mit den gleichen Daten trainiert, bis alle Skalierungswerte in einem Label Noise-Bereich durchgelaufen sind. Anschließend wird ein neuer MNIST Datensatz mit Label Noise erzeugt, und auf diesem für jeden Skalierungswert ein Convolutional Neural Network trainiert. Dies erfolgt für alle Ansätze auf beiden Basisnetzen und für beide Arten von Label Noise.

Aus den so gewonnenen Daten werden Punkte, in denen das Label Noise für Auffälligkeiten sorgt, entnommen und anhand dessen weitere Netze genauer trainiert, um exaktere Daten generieren zu können. Als auffällige Bereiche gelten dabei zum Beispiel starkes Abflachen der Accuracy sowie wenn eine Accuracy, entgegen der Erwartung, weiter besonders stabil bleibt.
Bei diesen weiterführenden Experimenten wird jeder Skalierungswert auf einer Menge an Label Noise mehrfach wiederholt, damit ein Standard gebildet werden kann. Dieser statistische Mittelwert soll bei den Vergleichen der einzelnen Accuracys untereinander helfen.

4.2 Symmetrisches Label Noise

Wie bereits in Kapitel 2 angesprochen, wird das Label Noise für diese Experimente zufällig, klassenunabhängig und somit symmetrisch [13] erzeugt. Dazu werden zufällig ausgewählte korrekte Label in eine andere Ziffer abgeändert. Diese neue Ziffer ist nicht identisch mit der korrekten Ziffer und ebenfalls zufällig ausgewählt.
Anschließend wird ein neues, noch korrektes Label ausgewählt und der Vorgang wieder-

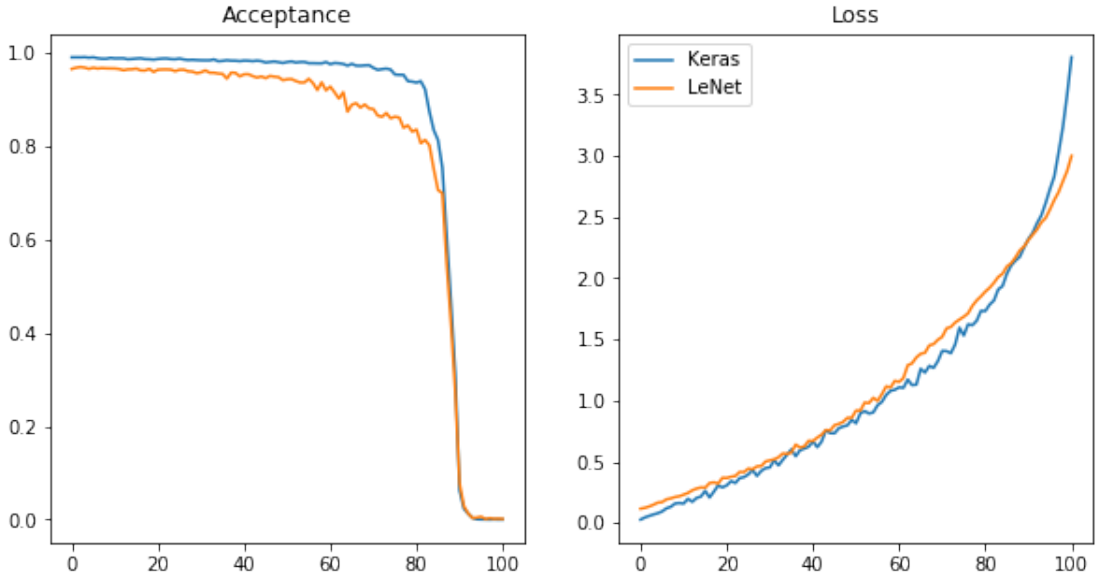

Abbildung 4.1: Accuracy und Loss der beiden Basisnetze, in Abhängigkeit zur Menge an Label-Noise in Prozent

holt, bis eine vorher definierte Menge an Labeln eine falsche Ziffer angeben. Auf diesen

veränderten Label weden die neuronalen Netze dann mit verschiedenen Skalierungen trainiert und auf nicht veränderten, korrekten Labeln getestet.
Die beiden Basisnetze gelten dabei als Grundlage, um eine mögliche Verbesserung in der Robustheit durch die drei Ansätze zu erkennen.

Dabei ist in Abbildung 4.1 für beide Basisnetze gut zu erkennen, dass ab ca. 80 Prozent Label Noise ein Lernen der korrekten Zuordnung schwer bis unmöglich zu sein scheint. Das LeNet fällt bei 74% Label Noise auf eine Accuracy von unter 84 Prozent und bei 90% Label Noise sogar auf unter zehn Prozent Accuracy. Das Keras-Netz dagegen hat eine von Grund auf um zwei Prozent höhere Accuracy. Anders als das LeNet flacht es, auch vor dem starken Abfall, ab ungefähr 85 Prozent Label Noise nicht merklich ab, stattdessen hat es noch über 92% Accuracy bei 82% Label Noise. Doch wie das LeNet fällt auch das Keras-Netz unter 10 Prozent Accuracy bei 90% Label Noise. Somit scheinen sowohl Werte im Bereich von wenig Label Noise bis 30 Prozent, wie es wahrscheinlich auch in der Realität vorkommt, als auch die Bereiche zwischen 70 und 90 Prozent Label Noise, in denen die Basisnetze stark nachlassen, als relevante Werte, für die eine Verbesserung der Accuracy interessant ist.

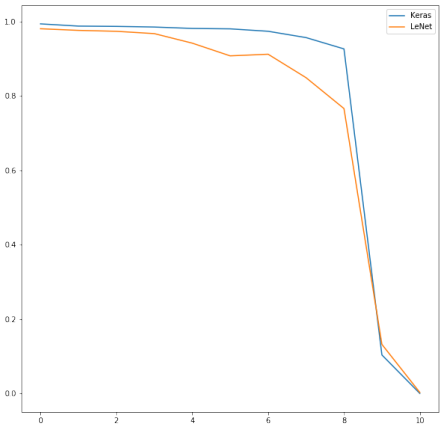

Abbildung 4.2: Accuracy der beiden um Skalilerung angepassten Netze, Skalierungswert von 1, in Abhängigkeit von Label Noise in Prozent

Scaled batch Norm Auch bei einer Veränderung der Basisnetze um den Ansatz der Scaled Batch Norm ist eine durch die Batch Normalization entstandene, bessere Accuracy von Keras, als auch LeNet zu erkennen. Dabei ist der Skalierungswert, der in Abbildung 4.2 dargestellten Werte, 1 und somit bleibt nur die reine Batch Normalization. Ohne Label Noise beträgt diese Differenz unter 1.5 Prozentpunkten. Das Abflachen der Kurven ist ähnlich zu den Standartnetzen, mit einer Accuracy von 92% bei Keras und 76% bei LeNet bei 80% Label Noise. Damit ist das LeNet in dieser Grundkonstellation leicht schlechter als der Ausgangswert des nicht veränderten Basisnetzes.

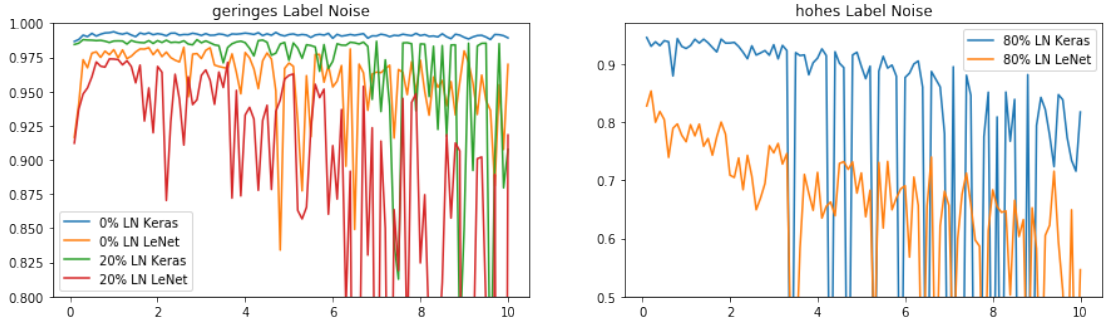

Abbildung 4.3: Accuracy in Abhängigkeit vom Skalierungswert, für verschiedene Mengen an Label-Noise

Bei Keras ist ohne Label Noise ein Höhepunkt der Accuracy bei einem Skalierungswert um 1 klar zu erkennen. Dieser Höhepunkt ist bei 20% Label Noise nicht mehr existent, stattdessen scheinen alle Werte eine ähnliche Accuracy zu erzielen. Bei hohem Label Noise, wenn die gesamte Accuracy, unabhängig des Skalierungswertes, geringer ist, wie oben und in Abbildung 4.2 beschrieben, kann die Accuracy durch einen geringen Skalierungswert von unter 2 verbessert werden.

Bei LeNet dagegen lässt sich bereits ohne die Einführung von Label Noise kein so ausgeprägter Höhepunkt der Accuracy wie bei Keras feststellen. Dennoch ist durch den minimalen Höhepunkt ein Einfluss des Skalierunsgwertes auf die Accuracy ersichtlich. Noch flacher ist der Höhepunkt bei 20% Label Noise, generell scheinen die Ergebnisse sich jedoch ähnlich zu verhalten. Bei 80 % Label Noise sind die Accuracy-Ergebnisse von LeNet, trainiert mit einem kleinem Skalierungswert, besser als bei hohen Skalierungswerten. Dennoch scheint der Einfluss der Skalierung auf Scaled Batch Normalization im Sinne der Robustheit gering zu sein.

Scaling mit Wurzel Bei der Veränderung der Basisnetze um den in Unterabschnitt 3.1.5 vorgestellten Ansatz der Skalierung mit anschließendem Radizieren, ist die Accuracy von LeNet, anders als in den bisher betrachteten Abbildungen, bei einem Skalierungswert von 1 um fast 5 Prozentpunkte geringer als die Accuracy des Keras-Netzes, noch bevor Label Noise eingeführt wird. Dafür wirkt LeNet in diesem Ansatz deutlich stabiler als z.B. bei der Scaled Batch Norm, mit einem Abfall von nur 7 Prozentpunkten Accuracy auf 88%, bei 70% Label Noise. Im gleichen Bereich fällt Keras um 4 Prozentpunkte, auf

95%Accuracy.

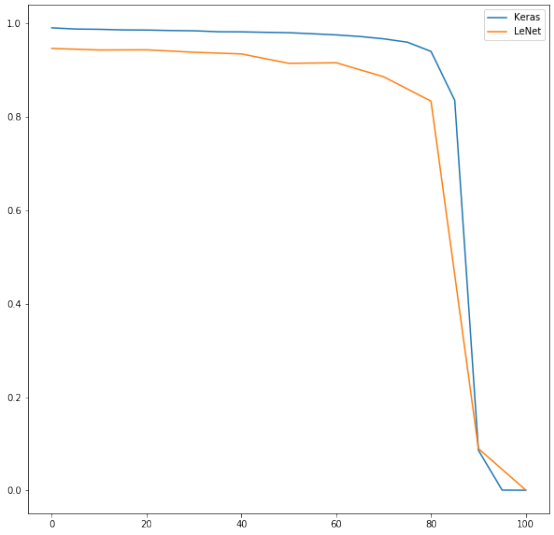

Abbildung 4.4: Accuracy der beiden um auf Ansatz 2 angepassten Netze, Skalierungswert = 1, in Abhängigkeit von Label Noise in Prozent

Bei beiden angepassten Basisnetzen sinkt die Accuracy bei 80 Prozent Label Noise weiter, bleibt jedoch über den Werten der unangepassten Basisnetze in Abbildung 4.1. Auch in diesem Ansatz sind die Werte ab 90% Label Noise ungenauer, als wenn immer nur eine Ziffer als Antwort angegeben wird.

Obwohl es für Keras aus Abbildung 4.5 nicht auf den ersten Blick ersichtlich wird, da die Differenz des besten und schlechtesten Wertes nur 0,005 Prozentpunkte beträgt, sorgt ein kleiner Skalierungsfaktor um circa 0,3 bei keinem oder geringem Label Noise für die höchste Accuracy. Dabei erreicht das Keras eine Accuracy von 99,2% ohne Label Noise und 98,9% bei 20 Prozent Label Noise. Interessanterweise fällt bei 80 Prozent Label Noise das Netz aus dem Muster, welches von 60, 70 und 90 Prozent Label Noise gezeichnet wird und hat einen Höhepunkt um einen Skalierungswert von 7.8. Die zuvor genannten Mengen an Label Noise dagegen erreichen die meiste Accuracy mit einer Skalierung um 2,4. Trotzdem ist das veränderte Keras-Netz in allen Bereichen besser als das Basis-Keras-Netz und kann damit eine Steigerung der Robustheit nachweisen.

Trotz dessen, dass das LeNet schlechter ist als Keras, kommt es bei 20 Prozent Label Noise mit einem Skalierungswert von 0,3 auf über 97% Accuracy und ist damit besser als das reine Basisnetz (in der Version LeNet) ohne Label Noise. Die höchste Accuracy erreicht das LeNet bei 80% Label Noise interessanterweise bei einem Skalierunsgwert von 1, welcher somit obsolet ist.

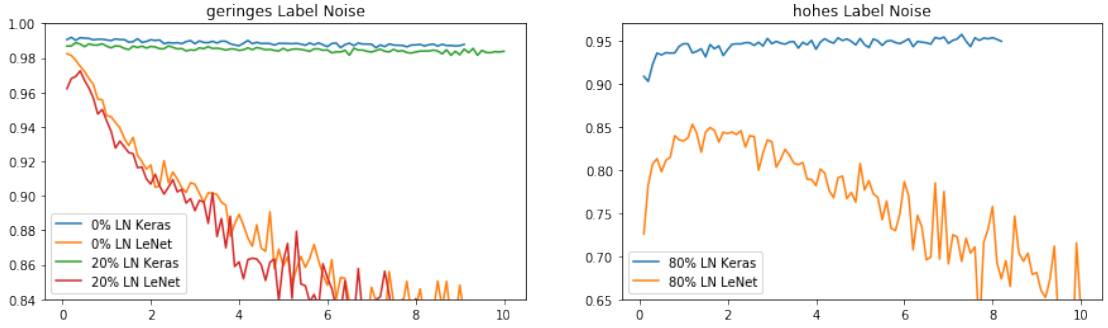

Abbildung 4.5: Accuracy in Abhängigkeit zum Skalierungswert, für verschiedene Mengen an Label Noise

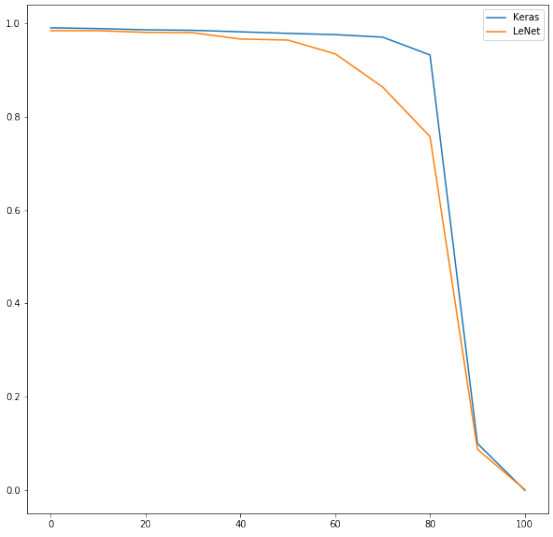

Abbildung 4.6: Accuracy der beiden um Skalierung angepassten Netze, Skalierungswert von 1, in Abhängigkeit von Label-Noise in Prozent

Reines Scaling In der dritten Veränderung der Basisnetze, durch Einfügen des Skalierungswertes vor der Softmax-Funktion, sind die Accuracy-Werte auf den ersten Eindruck dichter an den Werten der Scaled Batch Norm. Schließlich bleibt nicht nur das Keras-Netz, wie in Abbildung 4.6 zu erkennen, bei einem Skalierungswert von 1 bis 70% Label-Noise sehr stabil und verliert von den 99% Accuracy ohne Label Noise zwei Prozentpunkte. Auch 80% Label Noise sind noch bei über 90 % Accuracy. Darüber hinaus ist LeNet ohne Label Noise fast so akkurat wie das Keras-Basisnetz, mit knapp über 98% Accuracy. Bis 50 Prozent bleibt dieses ebenfalls stabil, flacht dann jedoch schneller als Keras ab und hat bei 80% Label Noise noch eine Accuracy von 75%.

Bei genauerer Betrachtung einzelner Label Noise-Bereiche über den Verlauf der Accuracy bei steigendem Skalierungswert ist, anders als bei der Scaled Batch Norm und ähnlich zum Scaling mit Wurzelziehen, zu sehen, dass beim LeNet ohne Label Noise eine leichte Verbesserung der Accuracy mit Skalierungswerten zwischen 0.2 und 2 erfolgt. Auch bei

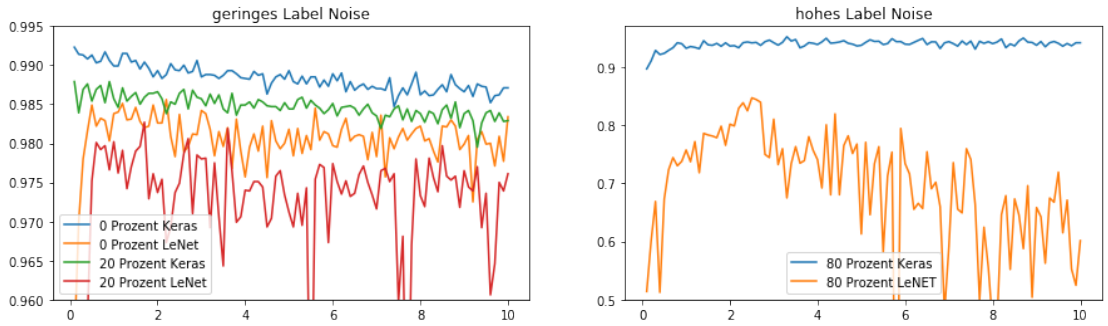

Abbildung 4.7: Accuracy in Abhängigkeit zum Skalierungswert, für verschiedene Mengen an Label-Noise

Keras trägt ein kleinerer Skalierungswert ebenfalls zu höherer Accuracy bei. Sowohl für LeNet als auch für Keras ist somit eine Verbessserung der Accuracy bei kleineren Skalierunsgwerten auch bei geringem Label Noise ersichtlich, bei Keras weiterhin ohne den Abfall der Accuracy durch zu kleine Skalierunswerte.
Anders verhält es sich bei 80% Label-Noise. Die höchste Accuracy kann bei dem LeNet mit einem Skalierungswert um 2,5 erreicht werden. Keras erreicht um 3,3 die meiste Accuracy, doch ist die Differenz in Accuracy, die durch Skalierung erzielt werden kann, geringer als bei LeNet.

4.3 Asymmetrisches Label Noise

Die zweite Art von Label Noise wird erstellt, indem jeder Ziffer eine andere Ziffer zugeordnet wird, dabei basiert die Zuordnung auf der Ähnlichkeit der Ziffern untereinander. Dadurch ist das Noise realitätsnaher als das bisher vorgestellte zufällig erzeugte Label Noise.
Da das Label Noise durch die direkte Zuordnung klassenbasiert und somit klassenabhängig ist, handelt es sich nach Patrini et. al.[13] um asymmetrisches Label Noise.

Zur Bestimmung, welche Ziffern vom genutzten Convolutional Neural Network verwechselt werden, wurde das LeNet Basisnetz auf korrekte Daten trainiert und anschließend beim Testen auf ebenfalls korrekte Daten die Wahrscheinlichkeitsverteilung der Ziffern

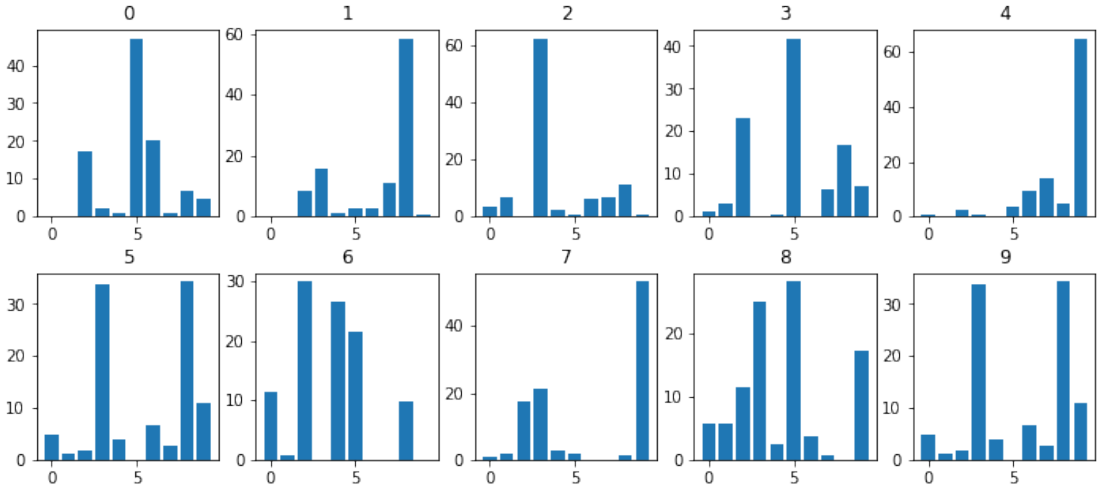

Abbildung 4.8: Wahrscheinlichkeitsverteilung ähnlich Zueinander aussehender Ziffern

zueinander abgelesen. Bei korrekter Klassifizierung wurde die zweitwahrscheinlichste Ziffer notiert. Dabei ergibt sich eine Häufigkeitsverteilung, abgebildet in Abbildung 4.8 von zweitwahrscheinlichsten und damit für diese Arbeit am ähnlichsten zueinander aussehenden Ziffern.

Wird also beim Erstellen des Label Noise eine Ziffer zufällig ausgewählt, wird diese, anders als bisher, nicht durch eine zufällige andere Ziffer ersetzt, sondern durch ihre in Tabelle 4.1 aufgeführte zweitwahrscheinlichste Partnerziffer.

korrekte Ziffer	zugeordnete Ziffer
0	5
1	8
2	3
3	5
4	9
5	8
6	2
7	9
8	5
9	8

Tabelle 4.1: Zuordnung der Ziffern zueinander

In Abbildung 4.9 ist der Unterschied zwischen dieser Art der Label Noise-Generierung und der vorherigen direkt erkennbar. Nicht nur sind Keras und LeNet ähnlich im Verlauf der sinkenden Accuracy mit steigendem Label Noise, auch die Kurve des Losses ist deutlich länger flach, bevor die Loss-Werte exponentiell steigen. Eine Erklärung für diese Art des Loss wird in Kapitel 5 gegeben.

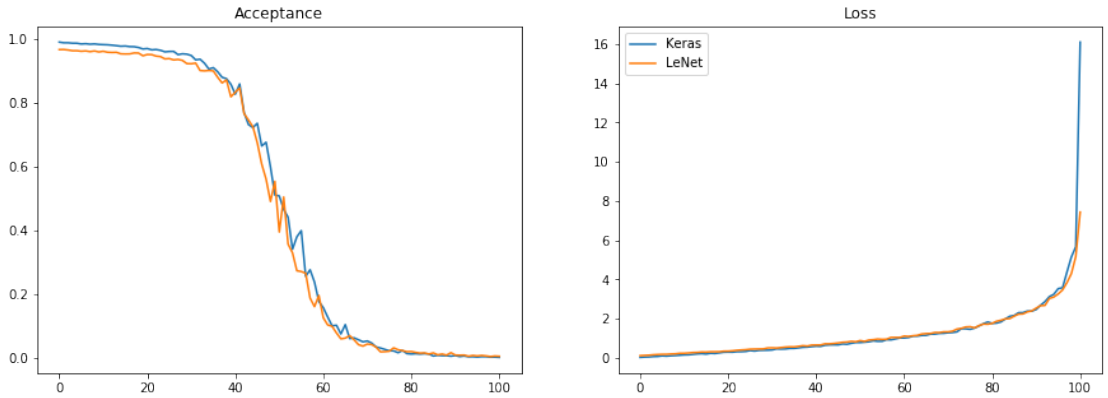

Abbildung 4.9: Accuracy und Loss der beiden Basisnetze, in Abhängigkeit zur Menge an Label-Noise in Prozent

Für den Vergleich der Accuracy ist besonders relevant, dass, auch wenn LeNet bis 30% Label Noise eine geringere Accuracy erreicht als Keras, bei steigendem Label Noise und der bei beiden stark fallenden Accuracy, die Werte der Accuracy beider Basisnetze ähnlich zueinander sind. Dies ist deutlich früher der Fall als beim symmetrischen Label Noise, bei gerade 60% Label Noise fallen beide Basisnetze unter 10 Prozent Accuracy. Daher wird der vorherige Bereich des hohen Label Noise dementsprechend von 70-90% auf 40-60% verschoben.

Scaled batch Norm Werden die beiden Basisnetze entsprechend dem ersten Ansatz um Skalierung und Batch Normalization erweitert, bleibt es dabei, dass LeNet und Keras bis 40 Prozent Label Noise einen ähnlichen Verlauf haben. Dabei ist LeNet im Falle der Scaled Batch Norm leicht stabiler, als ohne diese Veränderung und hat ohne Label Noise weniger als 2 Prozenpunkte, bei 20 Prozent Label Noise sogar nur 1 Prozentpunkt geringere Accuracy als Keras. Ab 40 Prozent Label Noise, wenn die Accuracy fällt, zeigt sich das LeNet sowohl eine bessere Accuracy als das zugehörige Basisnetz, als auch eine bessere Accuracy als Keras erzielen kann. Dennoch fallen beide Netze um 60% Label Noise auf unter 10% Accuracy.

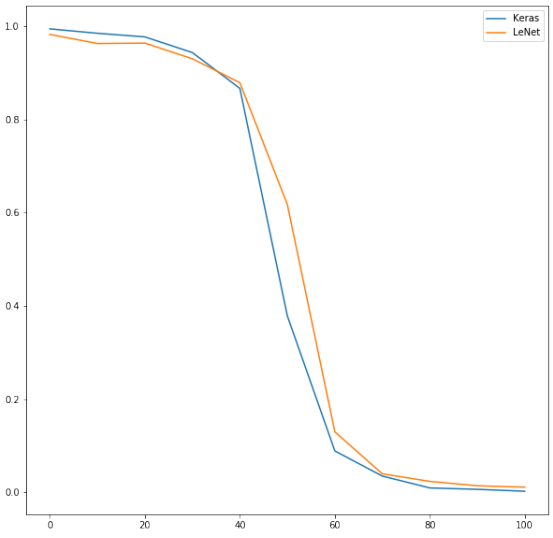

Abbildung 4.10: Accuracy der beiden um Scaled Batch Norm angepassten Netze, Skalierungswert von 1, in Abhängigkeit von Label Noise in Prozent

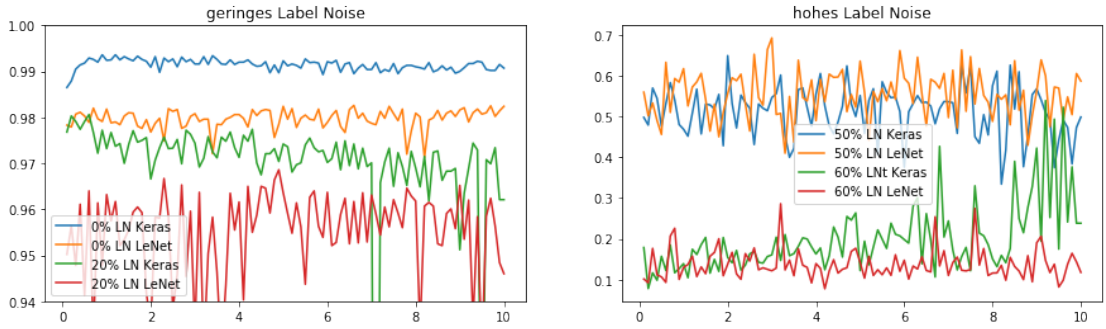

Abbildung 4.11: Accuracy in Abhängigkeit zum Skalierungswert, für verschiedene Mengen an Label Noise

In Abbildung 4.11 wird dabei deutlich, dass dieser aufgezeigte Verlauf nicht nur für ein Skalierungswert von 1 gilt. Sind die Skalierunsgwerte größer als 0.3, gibt es, außer bei 60% Label Noise beim Keras-Netz, keine ersichtlichen Höhe- oder Tiefpunkte. Alle Skalierungswerte zeigen Ausschläge, bleiben aber insgesamtgesehen konstant und verändern nichts an der Accuracy und somit der Robustheit des Convolutional Neural Networks.

Scaling mit Wurzel Mit nur einem Blick fällt, bei den um Skalierung und Radizieren erweiterten Basisnetzen, in Abbildung 4.12 mit einem Sklaierunsgwert von 1 abbgebildet, auf, dass besonders LeNet sich von den bisherigen Ergebnissen unterscheidet.

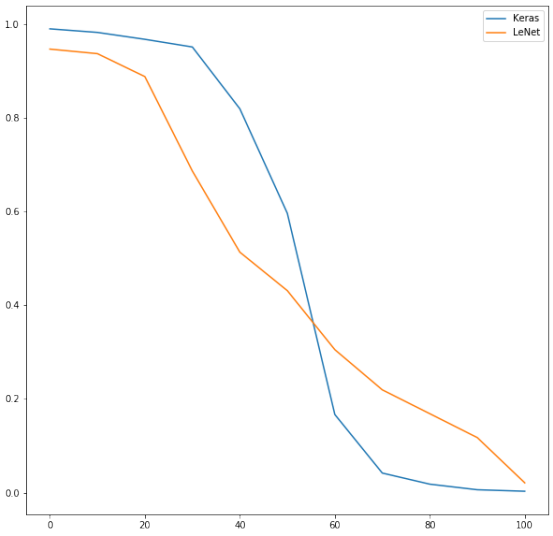

Abbildung 4.12: Accuracy der beiden um Scaled mit Wurzelziehen angepassten Netze, s = 1, in Abhängigkeit von Label Noise in Prozent

Dabei ist es reproduzierbar zu Beginn weniger akkurat als das Keras-Netz, und fällt anschließend ab 20% Label Noise fast linear in der Accuracy ab, statt wie allen anderen Graphen in einer Hyperbel.
Dadurch ist es ab 58% Label Noise in der Accuracy besser als das zugehörige Keras Netz und erreicht die 10% Accuracy Marke erst bei 90 Prozent Label Noise, ähnlich wie die auf der vorherigen Art von Label Noise getesteten Convolutional Neural Networks

Das Keras Netz ist leicht unter den Werten des Basisnetzes und fällt ab 40 Prozent Label Noise stark in der Accuracy, bis es bei etwas über 60% Label Noise die 10 Prozent Accuracy unterschreitet.

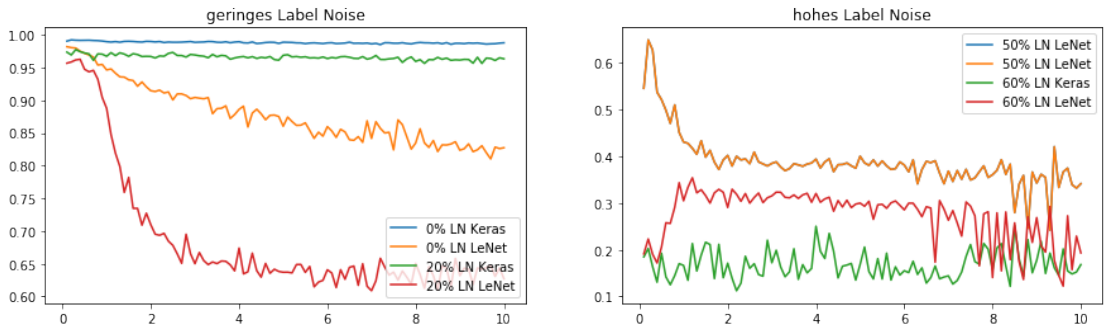

Abbildung 4.13: Accuracy in Abhängigkeit zum Skalierungswert, für verschiedene Mengen an Label Noise

Bei Keras ist, ähnlich wie bei der im vorherigen Paragraphen beschriebenen Scaled Batch Norm, kein Höhepunkt durch die Skalierung zu erkennen. Umso deutlicher sind dafür die Höhepunkte in der Accuracy bei LeNet. Bei 0 und 20 Prozent Label Noise, ebenso wie bei 50 und 60 Prozent Label Noise tragen Skalierungswerte zwischen 0.3 und 1.6 zu einer höheren Accuracy bei. Kleinere Skalierungswerte sind bei 0 Prozent Label Noise zwar ebenfalls gut, bei jeglicher Menge an Label Noise fällt die Accuracy bei zu kleinen Skalierunsgwerten jedoch.

Reines Scaling In der dritten Veränderung der Basisnetze, durch das Einfügen des Skalierungswertes vor der Softmax-Funktion, sind die Accuracy Werte von LeNet und Keras quasi identisch. Beide Netze erreichen eine Accuracy von knapp über 96% bei 20 Prozent Label Noise und fallen um 12 Prozenpunkte bis 40% Label Noise auf 84,2 (Keras) und 84,7 (LeNet) Prozent. LeNet ist bei hohem Label Noise leicht akkurater, beide Netze fallen aber knapp hinter 60% Label Noise unter 10 Prozent Accuracy.

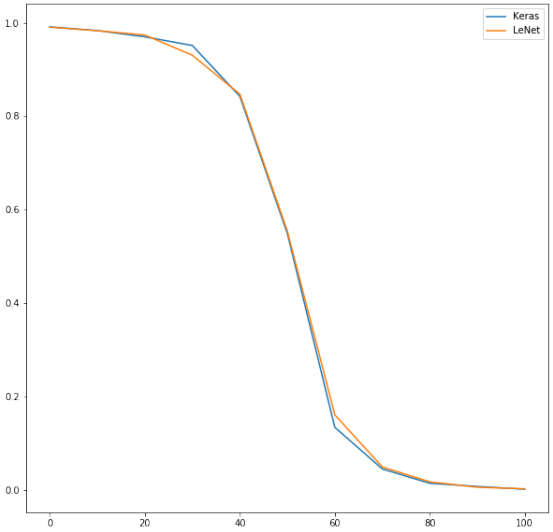

Abbildung 4.14: Accuracy und Loss der beiden um Skalierung ohne Radizierung angepassten Netze, Skalierungswert = 1, in Abhängigkeit zur Menge an Label Noise in Prozent

Da beide Netze sich ähnlich verhalten, ist LeNet leicht besser in der Accuracy als das reine Basisnetz, während Keras leicht schlechter ist.
Im Vergleich der Accuracy innerhalb einer Menge an Label Noise bei unterschiedlichen Skalierungswerten fällt noch einmal auf, wie ähnlich die beiden Netze sich verhalten.
Dabei sind im Bereich des geringen Label Noises leichte Verbesserungen der Accuracy für einen Skalierunsgwert von unter 2 zu erkennen, bei 20% Label-Noise ist die Auswirkung der Skalierung dabei prägnanter.
Bei hohem Label Noise ist ein Einfluss der Skalierung auf die Werte nicht mehr so offensichtlich. Kleinere Höhepunkte zwischen Skalierungswerten von 3 und 9 sind immer

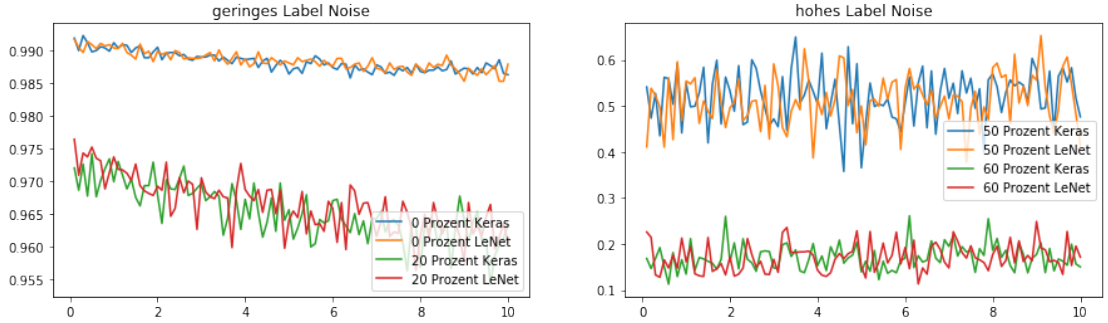

Abbildung 4.15: Accuracy in Abhängigkeit zum Skalierungswert, für verschiedene Mengen an Label Noise

wieder zu erkennen, doch können diese auch auf die Standardabweichungen der Netze zurückgeführt werden.

5 Auswertung und Diskussion

Im Folgenden werden die in Abschnitt 4.2 und Abschnitt 4.3 festgehaltenen Beobachtungen durch die bereits angesprochene genauere Betrachtung erweitert und interpretiert. Die Ergebnisse werden mit den Erwartungen aus Unterabschnitt 3.1.7 und den bekannten Erkenntnissen von [20] verglichen.
Anschließend werden Fehler der Arbeit aufgezeigt und die Durchführung diskutiert.

5.1 Auswertung

Zuerst wird individuell auf die Ergebnisse einer Art der Label Noise-Genierierung eingegangen. Anschließend werden die Ergebnisse aus beiden Varianten miteinander verglichen. Dabei sollen sowohl die Ansätze in sich evaluiert, als auch einander gegenübergestellt werden.

5.1.1 Symmetrisches Label Noise

Durch die in Abschnitt 4.2 vorgestellte Randomisierung des eingefügten Label Noise wird das Noise und damit das falsche Training über alle möglichen Ziffern verteilt. Es kann davon ausgegangen werden, dass diese Verteilung für die im Gegensatz zum asymmetrischen Label Noise, siehe Abschnitt 4.3, lange hoch bleibende Accuracy verantworlich ist. Wird zum Beispiel davon ausgegangen, dass das Random statistisch korrekt funktioniert und alle Ziffern gleich häufig zufällig ausgewählt werden, so würde die Ziffer 9 bei 70% Label Noise zu 30% korrekt klassifiziert gezeigt werden und zu 70% falsch klassifiziert. Diese 70% falsche Klassifikation teilt sich weiter auf, auf jeweils 7,78% der gezeigten Daten pro Ziffer, also zu 7,78% wird eine 1 als korrekt angenommen, zu 7,78% eine 2 und so weiter. Das bedeutet, das Netz bekommt eine korrekte Zuordnung zu einer Ziffer

viermal so oft gezeigt, wie zu einer spezifischen anderen, falschen Ziffer, obwohl nur grob ein Drittel der Ziffern korrekt gezeigt werden.
Das Netz ist sich weniger sicher in der einzelnen Klassifikation, wodurch der Loss steigt, kann aber im Training auf genügend Beispiele zurückgreifen, um noch lange einer, der korrekten, Ziffer gewogen zu sein.
Besonders der starke Abfall von über 90% Accuracy auf unter 10, zwischen 82 und 90 Prozent Label Noise sind mit dieser Theorie gut zu begründen.
Während bei 80% Label Noise das spezifische korrekte-zu-inkorrekte-Ziffer-Verhältnis noch bei 20 zu 8,88 Prozent liegt und damit eine korrekte Ziffer mehr als doppelt so häufig auch als korrekt belohnt, so ist dieses Verhältnis bei 90% Label Noise 10 zu 10 Prozent. Jede andere Ziffer wird dem neuronalen Netz so häufig als korrekt gezeigt wie eine tatsächlich korrekte.

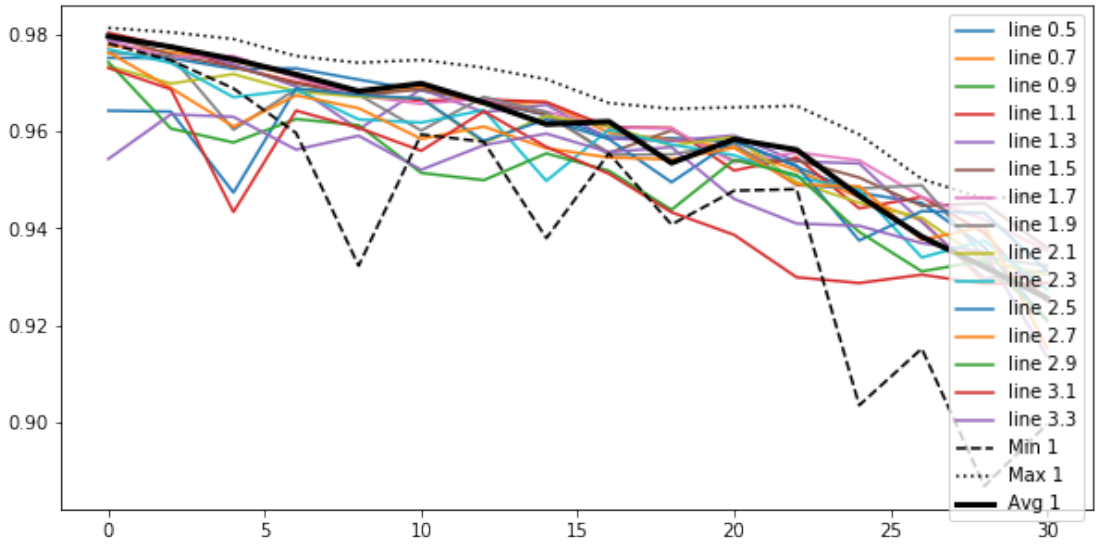

Abbildung 5.1: Accuracy von Scaled Batch Norm auf LeNet, in Abhängigkeit von Label Noise in Prozent

Scaled Batch Norm Auch wenn es in Abbildung 4.3 so aussah, als ob eine Skalierug mit kleineren Werten hilft, wird aus Abbildung 5.1 deutlich, dass diese Annahme auf zwei Irrtümern beruht. Zum einen ist der Höhepunkt identisch mit dem Höhepunkt bei einer Skalierung mit 1, zum anderen sind die Skalierungswerte, die außerhalb des für gut befundenen Bereiches liegen, deutlich schlechter als nicht skalierte Batch Normalization Accuracy-Werte.
Es wird deutlich, dass der Durchschnitt von zu kleinen Skalierungswerten, wie 0.1 im

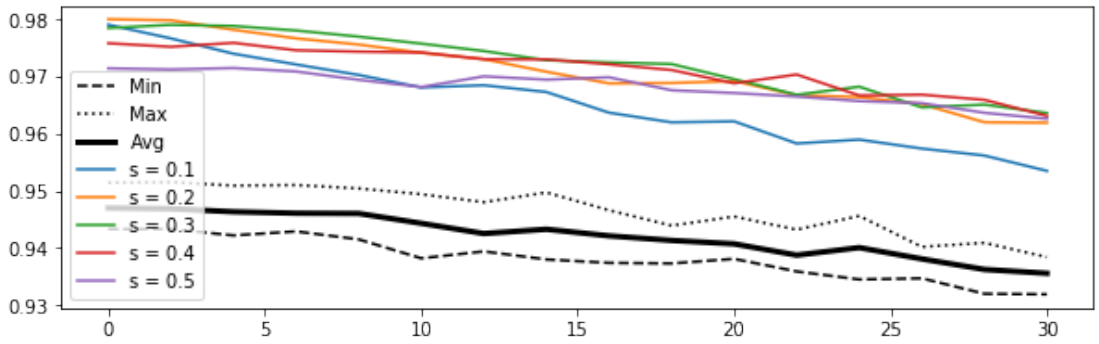

Abbildung 5.2: Accuracy von Skalieren und Radizieren auf LeNet, in Abhängigkeit von Label Noise in Prozent

Minimalbereich der Scaled Batch Norm mit dem Skalierungswert 1 liegen. Dieser Minimalbereich wird in Abbildung 5.1 durch eine schwarz gestrichelte Linie gezeigt und bildet die Minimalwerte von scaled batch norm mit s = 1 aus 10 Versuchen ab. Die schwarzgepunktete Linie am oberen Ende der Grafik stellt den Maximalbereich von s = 1 da. Alle andere Graphen sind mit den Mittelwerten über alle 10 Durchläufe erstellt.
Anders als s = 0,1 wirkt s = 0,9, wenn die Punkte dicht genug betrachtet werden, besser als s = 1. Dennoch unterschreitet es die Accuracy dieses Graphen manchmal und bleibt weit von der Maximalbereich Linie entfernt.
Insgesamt lässt sich somit feststellen, dass die Skalierung nur besser wirkt, wenn sich der Skalierungswert 1 annähert, da s = 1, also keine Skalierung, die beste Accuracy erzielt. Ansonsten verändert sich nur die Lernrate, um sich an die Skalierung anzupassen [17].

Scaling mit Wurzel Da die Auswirkungen des Skalierungswertes auf LeNet deutlicher sind als auf Keras, wurde sich auch in der genaueren Betrachtung und Auswertung, bei der Veränderung der Basisnetze um die Skalierung mit Radizieren, auf dieses beschränkt. Dieses Mal ist, anders als im vorherigen Ansatz, eine Auswirkung der Skalierung auch deutlich erkennbar. So sind die in Abbildung 5.2 abgebildeten Verläufe der Accuracy, von den Skalierungswerten 0,2; 0,3; 0,4; 0,5 alle um mindestens 3 Prozentpunkte besser als die Version mit einem Skalierungswert von 1. Ebenfalls abgebildet ist ein Skalierunsgwert von 0,9 als Vergleich zu einer zwar verbesserten, aber deutlich geringer verbesserten Robustheit.
Dass eine Skalierung um 0,3 besser ist als ein Skalierungswert von 0,2 oder 0,1, wobei s = 0,1 noch schlechtere Ergebnisse erzielt als s = 0,5, könnte an zwei möglichen Ursachen liegen. Zum einen ist es möglich, dass die Werte nach der Softmax-Aktivierung noch

mit der Wurzel des Skalierungswertes multipliziert und damit bei Skalierungswerten von 0,2 und 0,1 halbiert bzw. gedrittelt werden und somit zwar weiterhin die Reihenfolge beibehalten wird, aber der Effekt der vorherigen Skalierung zerstört wird. Zum anderen könnte es auch daran liegen, dass durch die große Skalierung ein Overfitting auf die Trainingsdatensätze entsteht.
Die Verbesserung im Bereich von hohem Label Noise ist deutlich marginaler. So können Skalierunsgwerte um 2 zwischen 75 und 84 Prozent Label Noise die Accuracy des LeNet um maximal 1,5 Prozentpunkte verbessern. Zudem sind die Skalierungswerte zu hoch, um vor 70% Label Noise einen positiven Einfluss auf die Robustheit des Convolutional Neural Networks zu haben.

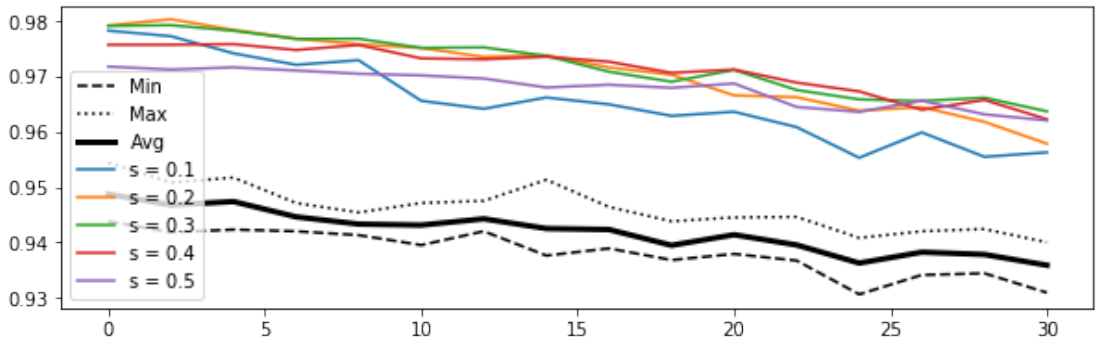

Abbildung 5.3: Accuracy von reinem skalieren auf LeNet, in Abhängigkeit von Label Noise in Prozent

Reines Scaling Im eigentlich nur als Vergleich gedachten, dritten Ansatz, in dem das anschließende Radizieren unterlassen wird, wird deutlich, dass bei den Bedingungen des Experimntes kein starker Vanishing Gradient Effekt auftritt. So sind die Ergebnisse fast identisch zu denen des vorherigen Ansatzes. Zum "Ausgleichen" des Radizieren bei Scaling mit Wurzel scheint ein Anpassen der Lernrate zu erfolgen, ansonsten hat dies keinen Einfluss.

Somit ist auch bei diesem Ansatz eine Skalierung mit Werten zwischen 0,2 bis 0,5 für geringes Label Noise besonders vorteilhaft und kann die Accuracy steigern. Ebenso ist die Auswirkung von Skalierungswerten auf die Accuracy bei hohem Label Noise gering und es können nur minimale Verbesserungen um bis zu 1 Prozentpunkt festgestellt werden.

5.1.2 Gegenüberstellung der Ergebnisse

Da das Problem der in Unterabschnitt 2.1.2 angesprochenen Vanishing Gradients nicht in dem für Ansatz eins und zwei benötigten Ausmaß besteht, ergibt es Sinn, dass die Skalierung bei der Scaled Batch Norm nicht den erwünschten Effekt gezeigt hat, da dieser Ansatz die Skalierung nur einsetzt, um Vanishing Gradients zu entgehen.
Somit ergeben sich auch die ähnlichen Ergebnisse der Ansätze zwei und drei, da das Radizieren des zweiten Ansatzes ebenfalls Vanishing Gradients vorbeugen soll.

In Abbildung 5.4 ist der Erfolg der Skalierung vor dem Softmax-Layer gut zu erkennen. So können Accuracy Werte ähnlich zu denen der Batch Normalization erreicht werden. Das Convolutional Neural Network wird also sowohl durch Skalierung vor dem Softmax-Layer, als auch durch Batch Normalization vor dem Softmax-Layer, welches im weitesten Sinne eine Art der Skalierung ist, gegen symetrisches Label Noise robuster werden.
Das liegt bei Ansatz zwei und drei hauptsächlich daran, dass durch die kleinen Skalierungswerte (die dividiert und nicht multipliziert werden) die Differenzen zwischen den einzelnen Logits vor dem Softmax-Layer bereits vertärkt werden, bevor der Skalierunsgeffekt des Softmax-Layers diese Differenzen weiter verstärkt. Somit werden die einzelnen Logits klarer voneinander abgegrenzt und das Convolutional Neural Network muss sich sicherer in seiner Klassifizierung werden, anstelle möglichst breit gestreut zu schätzen.

5.2 Asymmetrisches Label Noise

Das asymmetrische Label Noise wird erzeugt, in dem jeder Ziffer eine für das Convolutional Neural Network LeNet ähnlich aussehende Ziffer zugeordnet wird, dabei werden bei einem Prozentsatz der Label diese Ziffern vertauscht. Anders als beim symmetrischen Label Noise verteilt sich das Noise also nicht über alle inkorrekten Ziffern, die als korrekt vorgegeben werden, sondern nur auf zwei.

Das erklärt auch die Kurve des Loss in Abbildung 4.9. Anstelle zwischen allen zehn Klassen verwirrt zu werden, lernt das Netz, eine der beiden sich ähnlich sehenden Klassen als korrekt anzunehmen. Somit hält es diese bei einer Klassifizierung für wahscheinlicher, woduch die Differenz zwischen der vom Netz gegebenen Zahl zwischen 0 und 1 und der

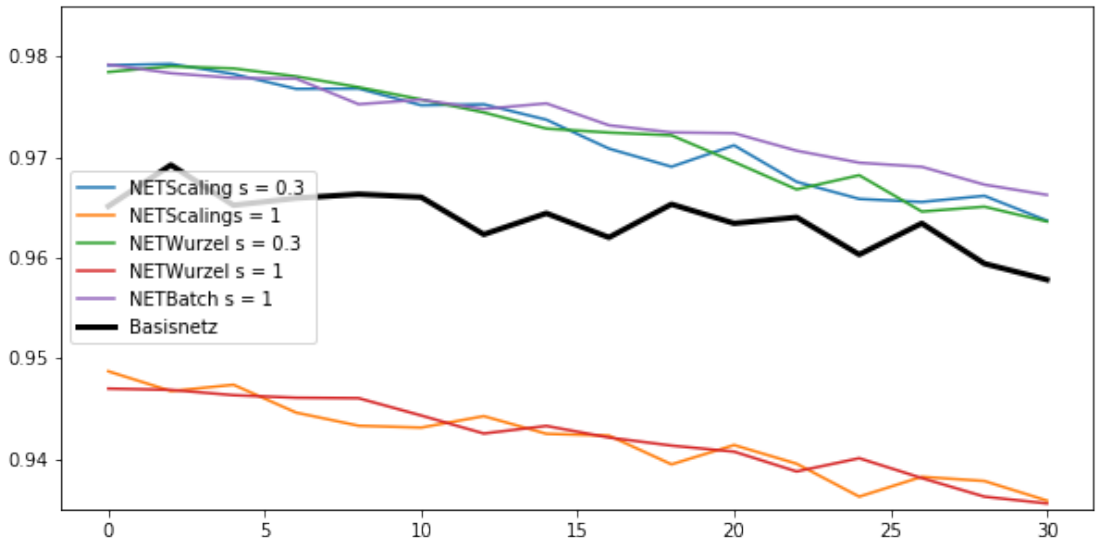

Abbildung 5.4: Genauere Betrachtung des geringen Label Noise Bereiches

korrekten Zahl, welche mit 1 angegeben wird, geringer ist. Außerdem sinkt die Differenz zwischen 0 und zumindest 8 der inkorrekten Ziffern. Diese Differenz wird, wie in Unterabsatz 3.1.1 erklärt, ebenfalls im Loss mitberechnet.
Auch der frühere Abfall der Accuracy, beziehungsweise die stärkere Empfindlichkeit gegenüber Label Noise lässt sich dadurch erklären. Generell ist es allgemein aktzeptiert, dass neuronale Netze gegenüber asymmetrischem Label Noise empfindlicher als gegen symmetrischem Label Noise sind [13].

Scaled Batch Norm Wie auch beim symmetrischen Label Noise kann der Effekt der Verbesserung der Accuracy durch Scaled Batch Norm nicht allgemein nachgewiesen werden. Wie in Abbildung 5.5 zu sehen, bleibt jegliche in Kapitel 4 "vielversprechend" aussehende Skalierung innerhalb des Minimal- und Maximalbereiches der einfachen Batch Normalisierung (s = 1). Durch eine Veränderung der Lernrate kann die Skalierung ausgegelichen werden, dies ist aber auch die einzige Veränderung [17].

Da das Paper von Zhu et. al.[20] die Skalierung aber hauptsächlich zur Bekämpfung des Vanishing Gradient-Problems vorschlägt und mit einer zehnfachen Epochengröße arbeitet, ist davon auszugehen, dass erneut "das Problem" der nicht vorhandenen Vanishing

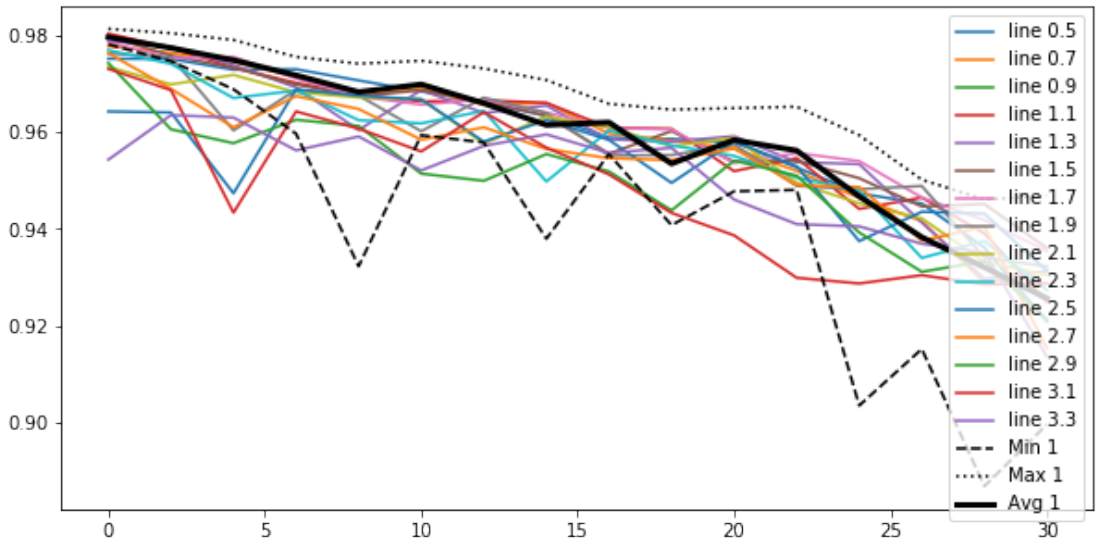

Abbildung 5.5: Accuracy von Scaled Batch Norm auf LeNet, in Abhängigkeit von Label Noise in Prozent

Gradients vorliegt. Der Nutzen von Scaled Batch Norm, im Sinne des Papers, kann dementsprechend nicht gegeben sein.

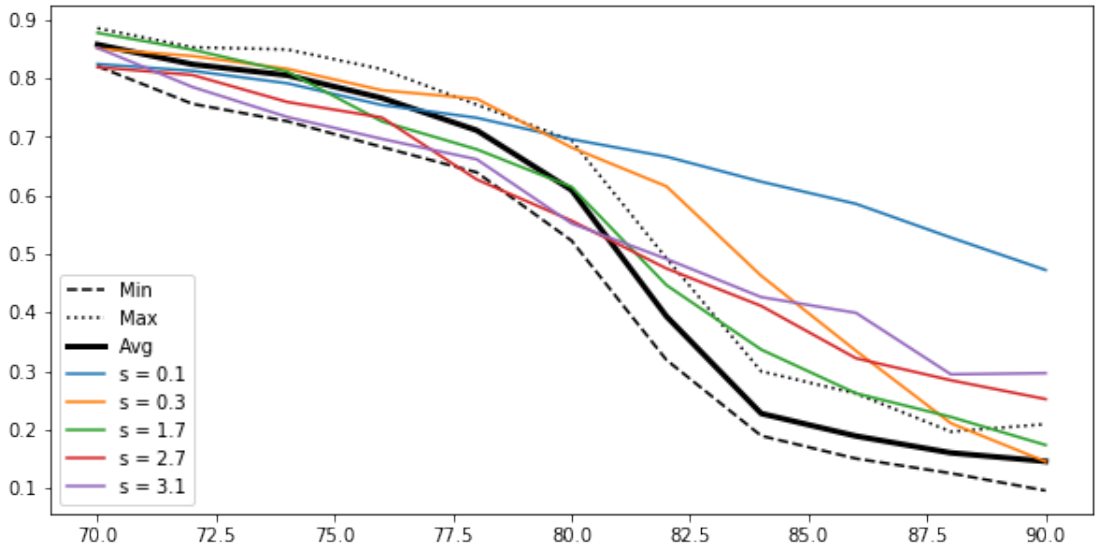

Abbildung 5.6: Accuracy von Scaled Batch Norm auf LeNet, in Abhängigkeit von Label Noise in Prozent

Eine Ausnahme dazu bildet nur der in Abbildung 5.6 dargestellte Bereich von hohem Label Noise bei LeNet, in dem eine Skalierung um 2,7 bis 3,3 eine Steigerung in der Accuracy um 15 Prozenpunkten bewirken kann. Da es sich dabei aber um eine Steigerung von 15 auf 30 Prozent Accuracy handelt, ist diese für realitätsbezogene Anwendungen eher zu vernachlässigen. Trotzdem kann somit die generelle Möglichkeit zur Steigerung der Robustheit von Batch Normalization durch Skalierung aufgezeigt und das Paper von Zhu et. al.[20] bestätigt werden.

Scaling mit und ohne Wurzel Im Folgenden geht es nun um die Ergebnisse der Skalierungen mit und ohne Wurzelziehen. Diese werden, durch die bereits mehrfach angesprochenen, nicht erreichten Vanishing Gradients und dem damit nicht benötigtem Radizieren, aufgrund ihrer fast identischen Resultate zusammen betrachtet.

Das Skalieren verhält sich, ebenso wie Scaled Batch Norm, quasi identisch zu der Version mit symmetrischem Label Noise. So ist bei Keras keine Verbesserung der Accuracy durch Skalierung zu erkennen, bei LeNet im geringen Label Noise Bereich dagegen schon.
Auch hier erreichen kleine Skalierungswerte zwischen 0,2 und 0,5 die beste Accuracy, doch können sie die Robustheit des Convolutional Neural Networks gegen Label Noise mehr verstärken, als beim Symetrischen Label Noise. Durch die Skalierung ist bei 30% Label Noise eine Verbesserung von 20 Prozentpunkten auf über 90% Accuracy möglich. Dabei ist ebenfalls auffällig, dass ohne Radizieren 0,2 ein noch effektiverer Skalierungswert zum Steigern der Accuracy ist, sodass durch das Radizieren wahrscheinlich die zuvor vergrößerten Werte zu stark minimiert werden.

Schließlich wird vor dem Softmax-Layer durch den Skalierungswert, ein Wert kleiner 1, dividiert und anschließend mit der Wurzel des Skalierungswertes, weiterhin ein Wert kleiner 1, multipliziert. Der Ausgabewert des Convolutional Neural Networks wird also erst vergrößert, dann werden diese skalierten Werte in einen Wahrscheinlichkeitsvektor umgerechnet, bei dem die Abstände durch die Mischung aus Skalierung und Softmax weiter maximiert und anschließend minimiert werden. Es ist dementsprechend folgerichtig nur passend, dass ein kleinerer Wert ohne Radizieren besser ist als dieser mit Radizieren, obwohl die Lernrate sich angepasst hat.
Im Gegensatz dazu wird bei einem Skalierungswert von 0,1, welcher um ein doppeltes

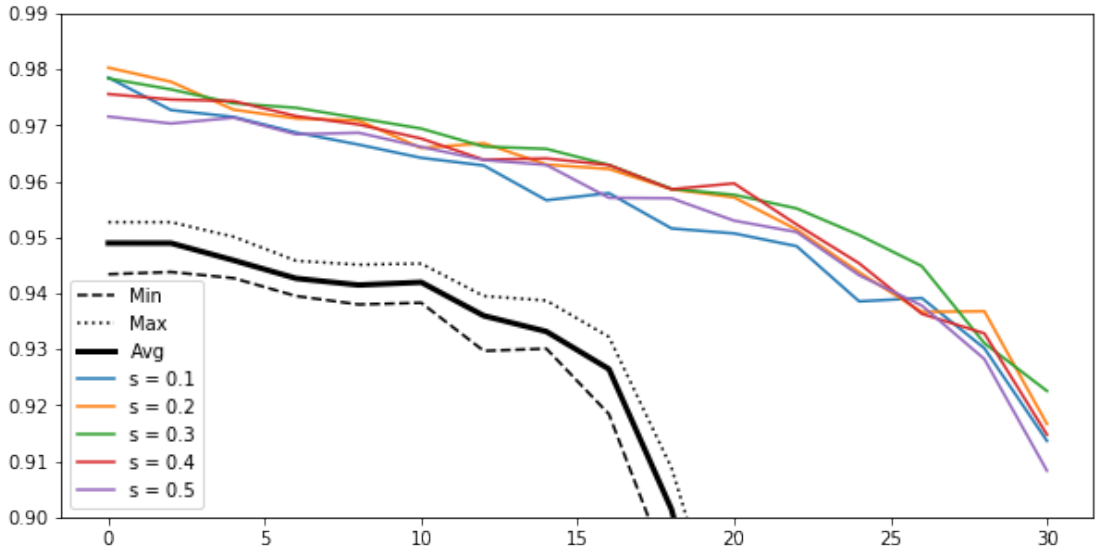

Abbildung 5.7: Accuracy von Skalieren auf LeNet, in Abhängigkeit von Label Noise in Prozent

Größer skaliert als 0,2, während des Trainings ein Overfitting entstehen, wodurch die Accuracy auf den Testdaten geringer ist

5.2.1 Gegenüberstellung der Ergebnisse

Wie in Abbildung 5.8 zu sehen, wird beim Vergleich der Ergebnisse deutlich, dass die Skalierung vor dem Softmax-Layer zu einer höheren Accuracy, im Ausmaß der Batch Normalization verhilft. Dabei ist der große Unterschied in der Accuracy zwischen dem reinen Scaling mit dem Skalierunsgwert s = 1 und dem Basisnetz, welche identische Ergebnisse abliefern müssten, auf eine Differenz der Trainingsserver und die Tatsache, dass Neuronales Netz nicht exakt reproduzierbar trainiert werden können, zurückzuführen.

Zur Gewinnung eines besseren Verständnisses, wieso die Skalierungen nur bei LeNet zu einer höhren Accuracy beitragen, wurden einige Experimente mit Hybriden aus beiden Basisnetzen durchgeführt. Dabei konnte aufgezeigt werden, dass nicht das in Unterabschnitt 3.1.7 angesprochene Dropout-Layer, sondern die Aktivierungsfunktion ReLu einen Erfolg der Skalierung unterdrückt.

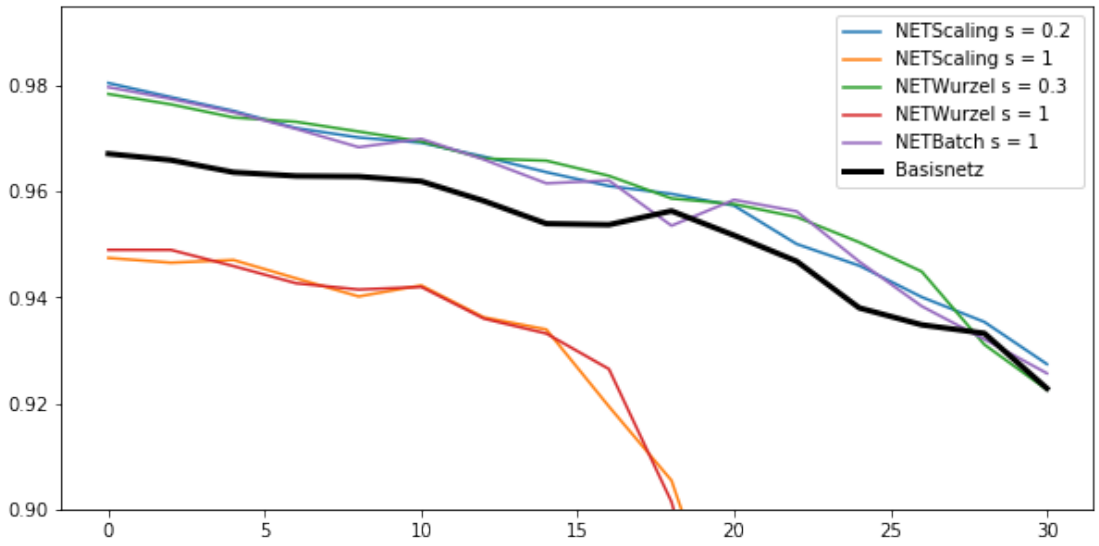

Abbildung 5.8: Genauere Betrachtung des geringen Label Noise Bereiches

In Abbildung 5.9 sind diese zwei Varianten der Anpassungen abgebildet. Dazu wurde einmal jede tanh Aktivierungsfunktion gegen ReLu ausgetauscht und bei dem anderen Netz ein Dropout-Layer an gleicher Stelle wie bei Keras eingefügt. Das LeNet mit Dropout Layer zeigt weiter eine Steigerung der Accuracy durch Skalierung, zudem sind die ohne Dropout trainierten Keras-Netze in der Accuracy gesunken, ein Skalierungseffekt konnte dennnoch nicht beobachtet werden.
Das LeNet mit ReLU Aktivierungsfunktionen dagegen zeigt ähnliche Skalierungseffekte wie Keras normalerweise.

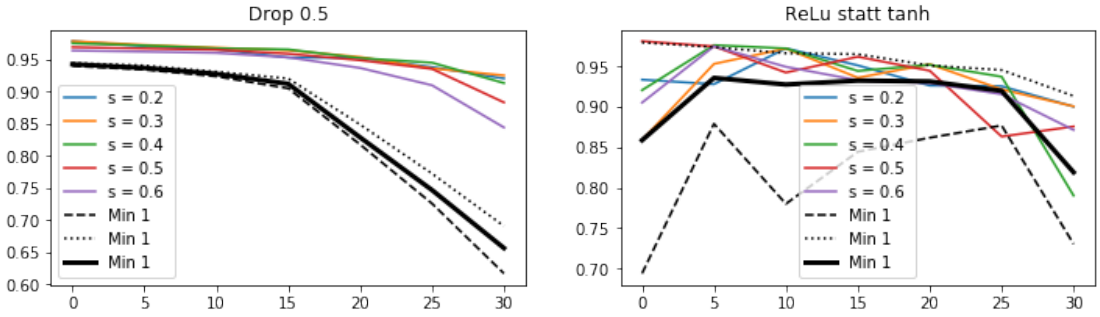

Abbildung 5.9: LeNet mit zwei verschieden Anpassungen

Unter den vorgestellten Betrachtungen wurde das reine Scaling auf LeNet mit einer Veränderung erneut trainiert. Anstelle durch den Skalierungswert zu dividieren, wurde mit diesem multipliziert. Dadurch ist der Bereich von bisherigen Skalierungswerten zwischen 0 und 1 einfacher zu betrachten, da dieser auf die Skalierunsgwerte zwischen 1 und 10 abgebildet wird.

Dabei sind drei Erkenntnisse hervorzuheben. Erstens ist bei wenig Label Noise 0,1 ein guter Skalierungswert, die Probleme des Overfittings entstehen erst durch Label Noise. Zweitens ist das Problem des Overfittings für einen zu kleinen, beziehungsweise in diesem Falle zu großen, Skalierungswert bestätigt worden und drittens wird deutlich, dass, sobald die Menge an inkorrekten Labels die Menge an korrekten übersteigt, ein kleiner, ehemals größerer Skalierungswert der Accuracy hilft. Besonders Letzteres ist gut in Abbildung 5.10 zu sehen, da bei 60% Label Noise die lokalen Minima in lokale Maxima umschlagen. Die Abbildung zeigt dabei immer einen Skalierungswert von 0,1 (ehemals 10) bis 10, (ehemals 0,1) durchskaliert auf 0, 10, 20 ... Prozent Label Noise, was die stufenhafte Abbildung ergibt.

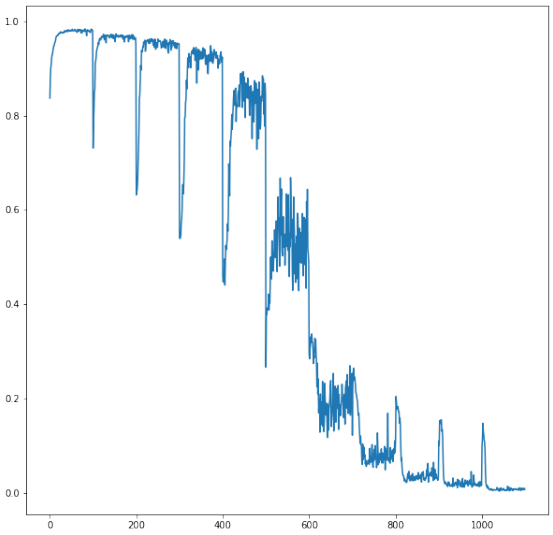

Abbildung 5.10: Accuracy von Skalieren auf LeNet, in Abhängigkeit von Label Noise in Prozent

5.3 Diskussion

Nachdem die verschiedenen Ansätze ausgewertet worden sind, folgt nun eine Diskussion der Ergebnissen der Arbeit. Es werden der Lösungsweg zur Fragestellung und das Ergebnis aus den Experimenten betrachtet. Dazu werden jeweils Vor- und Nachteile sowie potentielle Lücken aufgezeigt.

Betrachtung des Lösungsweg Zum Untersuchen der Auswirkungen von den drei ausgewählten Ansätzen ist ein Experimentieren und Austesten die vorteilhafte Vorgehensweise. Allerdings hätte schon früher auf Probleme mit den einzelnen Ansätzen eingegangen werden sollen, nicht erst beim Auswerten. Dadurch wurden aus Zeitgründen genauere Untersuchungen erschwert.
Besonders der dritte Ansatz des reines Skalierens ist als positiv hervorzuheben, da durch diesen erst die Probleme der ersten beiden Ansätze erkannt wurden.
Jedoch waren die Versuche zu breit aufgestellt, um Spezifika genauer betrachten zu können, das hindert die Auswertung.

Auch wenn die für das Training genutzten Server nicht auf eine Arbeit mit neuronalen Netzen ausgelegt sind, so ist der parallelisierte Ansatz bei den Experimenten von Vorteil. Der Vergleich von nicht exakt reproduzierbaren Ergebnissen aus dem Training der neuronalen Netze wäre sonst noch weiter erschwert worden.

Ungünstig ist der Ansatz, alle drei Veränderungen gegen sich selbst zu vergleichen. Es hätte immer ein Basisnetz als Maßstab mit trainiert werden sollen.

Betrachtung der Ergebnisse Die gewonnen Ergebnisse liefern einen guten ersten Ansatz zum Aufbau weiterer, tiefergehender Experimente. Bei denen zum Beispiel der dritte Ansatz der reinen Skalierung, aber auch die ersten beiden Ansätze in eienr Umgebung mit mehr Vanishing Gradients, untersucht werden könnten.
Da jedoch die Ergebnisse aus dem Paper von Zhu et. al[20] nur unter einer spezifischen Bedingung reproduziert werden konnten und die Ergebnisse auch nur in Teilen mit denen von Patrini et. al. [13] übereinstimmen, kann ein Fehler im Lösungsansatz sowie der Programmierung nicht ausgeschlossen werden. Diese Möglichkeit des Ausschlusses wäre aber für den weiteren Bezug auf diese Arbeit relevant.
Zudem sind alle Ergebnisse auf manuell eingefügtem, nicht natürlichem Label Noise erzielt und mit MNIST ein kleines Datenset eingesetzt worden, sodass der Realitätsbezug nicht vollständig gegeben ist. Diese beiden Punkte sollten für weitere Arbeiten definitiv verbessert werden.

6 Fazit und Ausblick

Im Folgenden werden die aus der Bachelorarbeit gewonnen Erkentnisse kurz zusammengefasst. Des Weiteren wird ein Ausblick auf weitere Untersuchungen gegeben, die auf Grundlage der Arbeit durchgeführt werden können.

Der Ausgangspunkt der vorliegenden Arbeit war die in Abschnitt 1.1 gestellte Frage, ob qualitativ schlechte Trainingsdaten durch eine Veränderung der Topologie ausgeglichen werden können. Nach Durchführung der beschriebenen Experimente lässt sich diese abschließend mit einem "teilweise schon" beantworten.
Auch wenn, wie in vielen Bereichen der Wissenschaft, keine definite Antwort möglich ist, so lassen sich definiv Verbesserungen der Accuracy durch Skalierung vor dem Softmax-Layer erkennen. Das Ganze bleibt aber eingeschränkt durch die Art und Menge an Label Noise. Auch ist in dieser Arbeit sehr deutlich geworden, dass die Topologie eines Neuronales Netzs sehr allgemein gehalten ist, so müssen Convolutional Neural Networks maximal in einer definierenden Aussage identisch sein, da alle eine Faltungsschicht beinhalten müssen. Doch Aktivierungsfunktionen, Tiefe des Netzes, Pooling und andere Layer können genug Einfluss auf die Topologie ausüben, um die Verbesserung der Accuracy durch Skalierung vor der Softmax-Funktion zu relativieren.

Dennoch ist eine Verbesserung der Accuracy und damit eine Steigerung der Robustheit des Convolutional Neural Networks eindeutig möglich. Auch wenn die Steigerung bei bereits akkurat klassifizierten Netzen teilweise sehr marginal ist, so ist eine Steigerung von 15 auf über 30% Accuracy durchaus beachtlich (Scaled Batch NBorm, LeNet, 60% Label-Noise).

Aus der erfolgten Untersuchung lassen sich darüber hinaus Ansätze für eine weiterführende Forschung ziehen: Zum einen könnte bewusst versucht werden, einen Vanishing Gradient zu erzeugen, um den Ansatz des Radizieren zu Überpüfen. Zum anderen könnte, wenn das Radizieren nicht benötigt wird, ein Skalieren mit negativen Werten ausgetestet

werden.
Ebenso sind alle drei Ansätze nur auf einem sehr kleinen Datensatz getestet worden und ein Testen auf zum Beispiel CIFAR100, wie in der Grundlage für Scaled Batch Norm, könnte ausprobiert werden.

Literaturverzeichnis

[1] Duden. https://www.duden.de/rechtschreibung/robust, 2022.

[2] Oxford languages. https://languages.oup.com/, 2022.

[3] Sunitha Basodi, Chunyan Ji, Haiping Zhang, and Yi Pan. Gradient amplification: An efficient way to train deep neural networks. *Big Data Mining and Analytics*, 3(3):196–207, 2020.

[4] C. E. Brodley and M. A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, aug 1999.

[5] F. Chollet. convnet mnist. https://github.com/keras-team/keras-io/blob/master/examples/vision/mnist_convnet.py, 1015. Accessed on 2022-07-17.

[6] F. Chollet. *Deep Learning with Python, Second Edition*. Manning, oct 2021.

[7] F.Tenzer. Volumen der jährlich generierten/replizierten digitalen datenmenge weltweit in den jahren 2012 und 2020 und prognose für 2025. https://de.statista.com/statistik/daten/studie/267974/umfrage/prognose-zum-weltweit-generierten-datenvolumen/, 2021.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[9] Alex Krizhevsky. Cifar100. https://www.cs.toronto.edu/~kriz/cifar.html. Accessed on 2021-05-12.

[10] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 12 1989.

[11] MNIST. Mnist. https://deepai.org/dataset/mnist. Accessed on 2021-05-12.

[12] B.Frénay M.Verleysen. Classification in the presence of label noise, a survey. *IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS*, 2014.

[13] Giorgio Patrini, Alessandro Rozza, Aditya Krishna Menon, Richard Nock, and Lizhen Qu. Making deep neural networks robust to label noise: A loss correction approach. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2233–2241, 2017.

[14] G. Press. Cleaning big data: Most time-consuming, least enjoyable data science task, survey says. https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/?sh=704a76356f63, 2021. Accessed on 2022-03-11.

[15] F.D. Ramos. Real-life and business applications of neural networks. https://www.smartsheet.com/neural-network-applications, 2021.

[16] Thomas A. Runkler. *Data Mining*. Vieweg+Teubner Verlag Wiesbaden, 2010.

[17] P. Stelldinger. Persönliches Gespräch. Geführt im Rahmen der Bachelorarbeit.

[18] P. Stelldinger. Master vorlesung zu helerfunktionen. Persönlich zur Verfüung gestellt. 2020.

[19] S. Ioffe C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *dblp*, 2015.

[20] Q.Zhu Z.He T.Zhang W.Cui. Improving classification performance of softmax loss function based on scalable batch-normalization. *Applied Sciences*, 2020.

[21] Can Yavuz. *Machine Bias Artificial Intelligence and Discrimination*. PhD thesis, 2019.

A Anhang

Alle verwendeten Skripte und generierten Daten werden zusammen mit einer CD/DVD abgegeben.

A.1 Code Beispiele

Der folgende Code wurde für die Basisnetze genutzt

Keras

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes),
        layers.Activation(activation= "softmax")
    ]
)
```

LeNet

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(6, kernel_size=(5, 5), activation="tanh",
         padding="same"),
        layers.AveragePooling2D(pool_size=(2, 2),
            strides=(1, 1), padding='valid'),
        layers.Conv2D(16, kernel_size=(5, 5), strides=(1, 1),
         activation="tanh", padding="valid"),
        layers.AveragePooling2D(pool_size=(2, 2),
         strides=(2, 2), padding='valid'),
        layers.Conv2D(120, kernel_size=(5, 5), strides=(1, 1),
         activation="tanh", padding="valid"),
        layers.Flatten(),
        layers.Dense(10, activation='tanh'),
        layers.Activation(activation= "softmax"),
    ]
)
```

Dabei wurde hinter Dense, anstelle des Bereits gegebenen Layers der Softmax-Aktivation angefügt:

Scaled Batch Norm

```
layers.BatchNormalization(),
layers.Lambda(lambda x: x * scaling_factor),
layers.Activation(activation= "softmax")
```

Scaling mit Wurzel

```
layers.Lambda(lambda x: x / scaling_factor),
layers.Activation(activation= "softmax"),
layers.Lambda(lambda x: x * math.sqrt(scaling_factor))
```

reines Scailing

```
layers.Lambda(lambda x: x / scaling_factor),
layers.Activation(activation= "softmax"),
```

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original