technische universität
dortmund

Texterkennung auf handgeschriebenen Dokumenten mit Long Short-Term Memory Networks

Nils Gerrit Brinkmann
14. August 2019

Betreuer:
Prof. Dr.-Ing. Gernot A. Fink
Eugen Rusakov, M.Sc.

Fakultät für Informatik
Technische Universität Dortmund
http://www.cs.uni-dortmund.de

INHALTSVERZEICHNIS

1 EINLEITUNG 3

2 GRUNDLAGEN 7

2.1 Grundbegriffe der Mustererkennung 7

2.2 Neuronale Netze 9

2.2.1 Training 10

2.2.2 Gradientenabstieg 11

2.2.3 Error Backpropagation 13

2.3 Faltungsnetze 15

2.3.1 Faltungsschichten 15

2.3.2 Pooling-Schichten 17

2.3.3 Voll-verbundene Schichten 17

2.3.4 Softmax-Schichten 18

2.4 Trainingsoptimierung 18

2.4.1 Überanpassung 18

2.4.2 Rectified Linear Units 21

2.4.3 Batch Normalisierung 23

2.5 Rekurrente Neuronale Netze 24

2.5.1 Long Short-Term Memory 26

2.5.2 Gated Recurrent Units 28

2.5.3 Bidirektionale rekurrente Netze 29

2.5.4 Multidimensionale rekurrente Netze 31

2.6 Connectionist Temporal Classification 32

2.6.1 Netzausgabe und Kostenfunktion 32

2.6.2 CTC Forward-Backward Algorithmus 34

2.6.3 Dekodierung 36

3 VERWANDTE ARBEITEN 37

3.1 Tiefe Faltungsnetze 37

3.2 NN-HMM Kombinationen 39

3.3 Attributraum-Repräsentationen 42

3.4 LSTM-CTC Kombinationen 46

3.5 Neuronale Netze mit Aufmerksamkeitsmechanismus 48

3.6 Augmentierung in der Handschrifterkennung 50

3.7 Class Activation Mapping 52

3.8 Diskussion 54
4 METHODIK 55
4.1 Convolutional Recurrent Neural Networks 55
4.2 Architektur von Joan Puigcerver 58
4.3 Architektur von Wigington et al. 61
4.4 Class Activation Maps für CRNNs 62
5 EVALUATION 63
5.1 Datensätze 63
5.1.1 IAM Handwriting Database 63
5.1.2 RIMES Database 64
5.1.3 George Washington Database 65
5.2 Metriken 67
5.3 Implementierung 69
5.4 Ergebnisse 71
5.4.1 Experimente auf dem IAM Datensatz 72
5.4.2 Experimente auf dem RIMES Datensatz 83
5.4.3 Experimente auf dem George Washington Datensatz 86
5.4.4 Vergleich der Architekturen 87
5.4.5 Class Activation Maps 89
6 FAZIT 93

1 EINLEITUNG

Das Lesen und Verstehen von Texten ist eine der grundlegendsten Fähigkeiten im Alltag eines Menschen, die bereits im Kindesalter erlernt wird. Für die Mensch-Maschine Interaktion ist es daher naheliegend, dass auch Maschinen Texte lesen und verstehen können sollen. Eine solche Interaktion kann beispielsweise auf einem berührungsempfindlichen Bildschirm erfolgen, auf dem eine Person mit dem Finger oder einem Stift schreibt. Auch die Durchsuchung und Analyse von archivierten Dokumentenbeständen oder das papierlose Büro sind Anwendungsszenarien, in denen das Verständnis vom Inhalt maschinell und handschriftlich erstellter Texte eine zentrale Rolle einnimmt. Um die gewünschte Reaktion der Maschine zu ermöglichen, muss das Geschriebene in einem ersten Schritt erkannt werden. In dieser Arbeit werden deshalb Methoden zur Erkennung handschriftlich erstellter Texte vorgestellt.

Für die Erkennung von maschinell erstellten Texten existieren bereits kommerzielle und Open-Source Werkzeuge wie OmniPage[1] oder OCRopus[2]. Die Erkennung von Handschrift ist hingegen heute noch ein aktives Forschungsthema, zu dem sich Wissenschaftler aus der ganzen Welt auf seit Jahren bestehenden Konferenzen wie der *International Conference on Frontiers in Handwriting Recognition* (ICFHR) oder der *International Conference on Document Analysis and Recognition* (ICDAR) treffen [Gue18, Bil17]. Die größte Schwierigkeit stellt die hohe Variabilität in der uneingeschränkten Handschrift zwischen verschiedenen Schreibern, aber auch beim mehrmaligen Schreiben eines Wortes durch den gleichen Schreiber dar.

Es existieren zwei grundlegende Ansätze für die Untersuchung von (handschriftlich verfassten) Textdokumenten. Die Texterkennung versucht ein Muster, wie eine gescannte Textseite, in Maschinenschrift zu überführen. Dadurch wird der gesamte Informationsgehalt des Textes in ein maschinenlesbares Format gebracht, und somit weitere Schritte wie die automatisierte Analyse des Inhalts mit Data-Mining Techniken erleichtert. Die korrekte Transkription handschriftlich erstellter Textdokumente stellt jedoch eine große Herausforderung dar. Das beste Verfahren von Forschern der RWTH Aachen in der *ICFHR2016 Competition on Handwritten Text Recognition on the READ Dataset* mit historischen Textdokumenten erzielte eine Wortfehlerrate von 20.9%

[1] siehe https://www.kofax.de/Products/omnipage
[2] siehe https://github.com/tmbdev/ocropy

[SRTV16]. Damit enthält im Durchschnitt etwa jedes fünfte Wort einen Transkriptionsfehler. Außerdem kann es insbesondere bei historischen Texten zu Uneinigkeit über die Bedeutung der Zeichen kommen, sodass die Definition der korrekten Transkription problematisch sein kann.

Eine Alternative zur Erkennung von Texten ist das sogenannte *Word Spotting*, dem anstelle der Transkription ein Retrieval Ansatz zugrunde liegt. Beim Word Spotting werden in einem Dokument ähnliche Vorkommen zu einer Anfrage (*Query*) gesucht. Im *Query-by-Example*-Ansatz entspricht die Anfrage einem Bildausschnitt und beim *Query-by-String* einer textuellen Repräsentation des Wortes [Sud18, S. 39f]. Word Spotting liefert im Vergleich zur Texterkennung robustere Resultate, weil eine nach Relevanz sortierte Liste von Vorkommen im Dokument statt der Transkription ausgegeben wird. Es kann vor allem unterstützend für den Anwender, beispielsweise bei der Durchsuchung von Dokumentenbeständen eingesetzt werden, bietet aber weniger Einsatzmöglichkeiten innerhalb voll-automatisierter Verfahren, wie der Sortierung von Post. Daher ist die Transkription in vielen Anwendungsfällen unumgänglich.

Die Texterkennung hat eine vielseitige methodische Geschichte. In den 1990-er Jahren waren Verfahren mit Hidden Markov Modellen verbreitet, die zuvor bereits in Spracherkennungssystemen eingesetzt wurden [SMSC94]. Früh wurden auch Neuronale Netze für die Merkmalsextraktion verwendet, deren Kombination mit Hidden Markov Modellen zur Segmentierung und Klassifikation als hybrider Ansatz bezeichnet wird. Die *Connectionist Temporal Classification* wurde im Jahr 2006 für die Spracherkennung entwickelt [GFGS06] und zwei Jahre später für die Handschrifterkennung adaptiert [GLF+08]. Sie ermöglicht den ausschließlichen Einsatz von Neuronalen Netzen zur Merkmalsextraktion, Segmentierung und Klassifikation von Handschrift innerhalb eines Netzdurchlaufs und liefert die methodische Grundlage für die besten Verfahren auf verschiedenen Datensätzen und Wettbewerben (siehe z. B. [SRTV16, MM19]). Auch mit der Connectionist Temporal Classification bleibt die Segmentierung von Zeilen und Wörtern zur Transkription einer Textseite ein Problem, das meist mit heuristischen Verfahren gelöst wird. Viele Datensätze wie die *IAM Handwriting Database* [MB02] stellen deshalb vor-segmentierte Wort- oder Zeilenabbilder zur Verfügung, an denen die Verfahren gemessen werden. Dies wird als wort- oder zeilenbasierte Transkription bezeichnet, die in Abbildung 1.0.1 dargestellt wird.

Ein zentraler Baustein für den Erfolg Neuronaler Netze innerhalb der Texterkennung ist der Einsatz von rekurrenten Neuronalen Netzen, insbesondere *Long Short-Term Memory Networks*. Die Kombinationen aus Faltungs- und rekurrenten Netzen ist in der Schrifterkennung erheblich leistungsfähiger als Faltungsnetze allein [MM19]. Die erzielten Ergebnisse sind auf die Möglichkeit der rekurrenten Schichten zurückzuführen, Informationen aus der Betrachtung des Kontextes eines Merkmals in der Klassifikati-

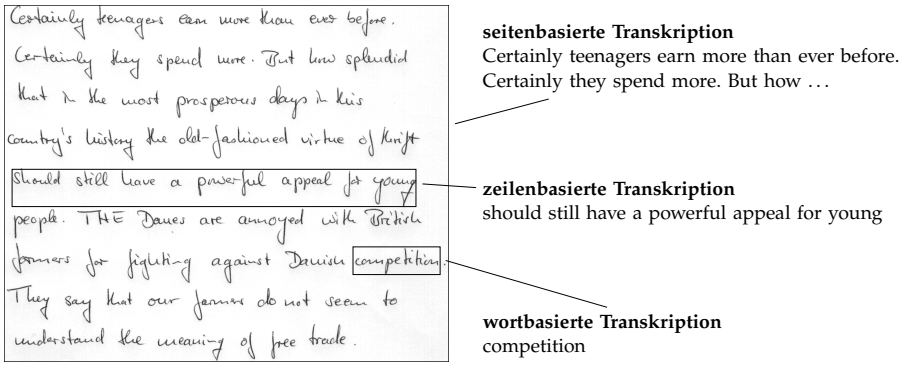


Abbildung 1.0.1: Darstellung der seiten-, zeilen- und wortbasierten Transkription anhand eines Beispiels aus der IAM Handwriting Database [MB02].

on einzubeziehen. Die numerische Instabilität des Trainings allgemeiner rekurrenter Schichten kann durch den Einsatz von Long Short-Term Memory Networks gelöst werden. Die Bezeichnung für diesen Typ rekurrenter Schichten gibt einen Hinweis auf die Fähigkeit, Informationen über einen langen Zeitraum zu speichern, sodass ein Muster in seinem gesamten Kontext betrachtet werden kann. Die Verbindung von Long Short-Term Memory Networks mit Faltungsschichten zu sogenannten *Convolutional Recurrent Neural Networks* (CRNNs) ist daher ein vielversprechender Ansatz für die Texterkennung. Innerhalb dieser Arbeit sollen zwei CRNN-Implementationen von Puigcerver [Pui17] und Wigington et al. [WSD+17] anhand mehrerer Datensätze getestet und miteinander verglichen werden. Das Ziel ist es, den Einfluss verschiedener Architekturentscheidungen auf die Transkriptionsleistung zu analysieren. Der folgende Abschnitt beschreibt, wie die dafür notwendigen Schritte in dieser Arbeit gegliedert sind.

GLIEDERUNG Kapitel 2 erklärt die fachlichen und methodischen Grundlagen für das Verständnis des verwendeten Texterkennungsverfahrens. Dazu zählt neben einer kurzen Einführung in die Mustererkennung und einer Erklärung von (rekurrenten) Neuronalen Netzen insbesondere die bereits erwähnte Connectionist Temporal Classification. Kapitel 3 zeigt eine Reihe verwandter Methoden, welche die Entwicklung der in Kapitel 4 vorgestellten CRNN-Architekturen beeinflusst haben. In Kapitel 5 wird das Evaluationsprotokoll und die Metriken beschrieben, welche die Grundlage für die Bewertung und den Vergleich verschiedener Texterkennungsverfahren bietet. Hier werden auch die Resultate der durchgeführten Experimente präsentiert. Kapitel 6 enthält eine abschließende Bewertung der Ergebnisse.

2 GRUNDLAGEN

In diesem Kapitel werden grundlegende Konzepte und Begriffe vorgestellt, die für das Verständnis der in dieser Arbeit angewendeten Methodik von Bedeutung sind. Abschnitt 2.1 erläutert die traditionelle Vorgehensweise von Mustererkennungssystemen und stellt die Arbeitsweise der Neuronalen Netze dieser gegenüber. Die Funktionsweise, der Aufbau und das Training von Neuronalen Netzen wird in Abschnitt 2.2 vorgestellt und Abschnitt 2.3 enthält eine Einführung in die für Bildverarbeitung besonders wichtigen Convolutional Neural Networks (CNNs). Für die Verarbeitung der Schriftbilder als zeitliche Sequenz werden in Abschnitt 2.5 die rekurrenten Neuronalen Netze präsentiert, die zusammen mit der in Abschnitt 2.6 vorgestellten Connectionist Temporal Classification die Grundlage für das zugrundeliegende Verfahren für die Handschrifterkennung bilden.

2.1 GRUNDBEGRIFFE DER MUSTERERKENNUNG

Bei der Mustererkennung geht es nach Niemann im weitesten Sinne um die mathematisch-technischen Aspekte von Perzeption, also um die Nachbildung der Wahrnehmungsleistung von Menschen oder Tieren durch Maschinen [Nie03, S. 10]. Die Bildverarbeitung und im Speziellen die Handschrifterkennung ist ein bedeutender Teilbereich der Mustererkennung. Der folgende Abschnitt orientiert sich, sofern nicht anders gekennzeichnet, an der Darstellung in [Nie03].

Klassische Mustererkennungssysteme haben im Normalfall einen mit dem Schema in Abbildung 2.1.1a vergleichbaren Aufbau. Der erste Schritt ist die Digitalisierung der physikalischen Größen durch Sensoren wie Mikrofone, Kameras oder Scanner. Im Falle der Handschrifterkennung werden normalerweise bereits vorhandene Dokumente gescannt, um ein digitales Bild zu erhalten. Alternativ wird eine zeitliche Abfolge von Koordinaten der Stiftspitze mithilfe eines berührungsempfindlichen Feldes aufgezeichnet. Die erste Variante wird als Offline- und die zweite als Online-Handschrifterkennung bezeichnet.

Innerhalb der Vorverarbeitung wird das aufgezeichnete Muster so transformiert, dass die weiteren Verarbeitungsschritte erleichtert werden. Dies wird vor allem durch das Verwerfen des für die Aufgabe irrelevanten Teils der Information und durch die

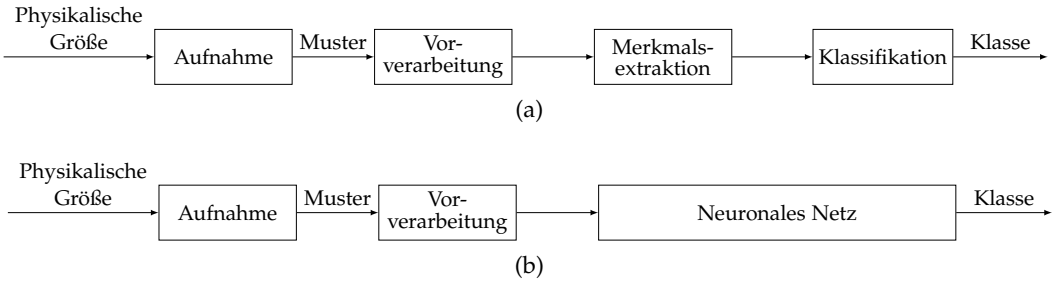


Abbildung 2.1.1: Pipelines (a) eines klassischen Mustererkennungssystems nach [Nie03, S. 26] und (b) eines Systems mit Neuronalen Netzen.

Reduktion von unerwünschter Variabilität in den Mustern erreicht. Unerwünschte Variabilität bezeichnet Unterschiede zwischen den Mustern einer Klasse, die durch die Vorverarbeitung reduziert werden können, um die Klassifikation zu erleichtern. Ein Beispiel hierfür ist die Konvertierung eines Farbbildes zu einem Graustufenbild. Bei Aufgabenfeldern wie der Handschrifterkennung, in denen die Schriftfarbe keinen Einfluss auf das geschriebene Wort hat, verringert ihre Entfernung die zu verarbeitende Datenmenge ohne die Qualität des Erkennungssystems negativ zu beeinflussen.

Auf die Vorverarbeitung folgt die Merkmalsextraktion, in der die vorverarbeiteten Muster in für die jeweilige Klasse möglichst spezifische numerische Merkmale überführt werden. Oftmals ist die Vorverarbeitung nicht klar von der Merkmalsextraktion getrennt und der Übergang nicht fest definiert. Zentral bei der Merkmalsextraktion ist die Reduktion der Datenmenge auf den für die Klassenunterscheidung erforderlichen Teil. Beispiele für Merkmale sind die *Histogram of Oriented Gradients* (HOG) Deskriptoren [DT05] oder geometrische Merkmale [WFS05], die ein kleines über das Eingabebild gleitendes Analysefenster im *Sliding-Window* Verfahren betrachten.

Der abschließende Schritt eines Mustererkennungssystems ist die Klassifikation, die die zuvor berechneten Merkmale und damit die zugehörigen Muster ihren jeweiligen Klassen zuordnet. Hierfür existiert eine Vielzahl verschiedener Methoden, wie die statistische Klassifikation, das Nächste-Nachbar Verfahren oder Support-Vektor Maschinen.

Alle Verfahren dieser Art haben gemeinsam, dass die Algorithmen von der Vorverarbeitung bis zur Klassifikation heuristisch ausgewählt werden. Das resultierende Problem ist, dass alle Teilschritte unabhängig voneinander an die vorliegenden Daten angepasst werden, diese jedoch bei der letztendlichen Klassifikation gemeinsam zum richtigen Ergebnis führen müssen. Im Falle klassischer Mustererkennungssysteme existiert keine Möglichkeit alle Teilschritte des Systems gemeinsam zu optimieren.

Dies ist ein entscheidender Unterschied zur Klassifikation mithilfe von Neuronalen Netzen [LBBH98].

Neuronale Netze ermöglichen eine integrierte Merkmalsextraktion und Klassifikation in einem Verfahren. Durch die große Modellkapazität und Generalisierungsfähigkeit der Netze wird oftmals auch der Umfang der Vorverarbeitung reduziert. Dies resultiert in einer Modellarchitektur wie sie in Abbildung 2.1.1b gezeigt ist. Der grundlegende Vorteil der Neuronalen Netze besteht darin, dass die verwendeten Merkmale nicht heuristisch festgelegt, sondern anhand eines Trainingsdatensatzes erlernt werden. Dies verringert den manuellen Aufwand für jeden Datensatz eine geeignete Menge von Merkmalen auszuwählen und ermöglicht das sogenannte Ende-zu-Ende Training. Innerhalb eines Trainingsverfahrens werden gleichzeitig sowohl die Merkmalsextraktion als auch der Klassifikator anhand des Trainingsdatensatzes optimiert.

2.2 NEURONALE NETZE

Künstliche Neuronale Netze sind aus einer Vielzahl von Neuronen aufgebaut, deren Struktur von biologischen Neuronalen Netzen wie dem menschlichen Gehirn inspiriert ist [MP43]. Jedes einzelne künstliche Neuron berechnet eine einfache mathematische Funktion $y = \Theta(\sum_i w_i x_i)$, die aus einer mit den Parametern w_i gewichteten Summe der Eingabewerte x_i und der Aktivierungsfunktion Θ besteht (siehe Abbildung 2.2.1).

Wichtige Anforderungen an die Aktivierungsfunktion sind die Nichtlinearität und die Differenzierbarkeit. Wird eine lineare Aktivierungsfunktion verwendet, können alle Schichten des Neuronalen Netzes zu einer einzigen linearen Funktion reduziert werden. Für das Gradientenabstiegsverfahren im Training wird eine Kostenfunktion (siehe Abschnitt 2.2.2) bezüglich aller Gewichte des Netzes differenziert, wofür alle Bestandteile des Neuronalen Netzes und damit auch die Aktivierungsfunktion differenzierbar sein müssen. Ein Beispiel für eine Aktivierungsfunktion ist die Sigmoidfunktion

$$\Theta_{\mathtt{Sigmoid}}(a) = \sigma(a) = \frac{1}{1+\exp(-a)}. \tag{2.2.1}$$

Weitere Aktivierungsfunktionen werden im Abschnitt 2.4.2 vorgestellt. Ein besonderer Parameter stellt der Bias dar, der in Abbildung 2.2.1 durch die Eingabe $x_0 = 1$ mit Gewicht w_0 dargestellt wird. Mit dem Bias kann eine feste Verschiebung der Eingabe in die Aktivierungsfunktion erreicht werden.

In Multi-Layer Perceptrons (MLPs) wird eine Vielzahl solcher künstlicher Neuronen in Schichten angeordnet, sodass die Ausgaben y_i^{n-1} in Schicht $n-1$ als Eingabe x_i^n aller Neuronen in Schicht n verwendet werden. Diese Schicht voll-verbundener Neuronen wird als *Fully-Connected Layer* bezeichnet. Die Anordnung der Schichten

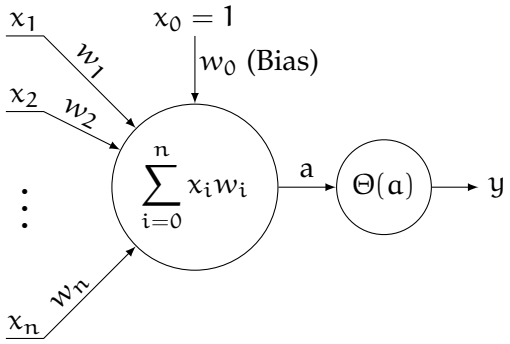


Abbildung 2.2.1: Ein Neuron berechnet die gewichtete Summe der Eingabewerte x_i mit den Parametern w_i. Diese wird durch die Aktivierungsfunktion Θ auf die Ausgabe y abgebildet. Grafik nach [Roj96].

hintereinander wird aufgrund der Richtung der Neuronenverbindungen *Feed-Forward Network* genannt. Ein solches Neuronales Netz ist bereits bei drei Schichten von Neuronen dazu in der Lage, beliebige Funktionen zu approximieren, sofern die mittlere Schicht über ausreichend Neuronen verfügt [HSW89]. Moderne Netzarchitekturen bestehen trotzdem oft aus einer wachsenden Anzahl an Schichten. Diese wird auch als Tiefe des Netzes bezeichnet und erklärt den Ursprung des Begriffs *Deep Learning*. Während die erste erfolgreiche Netzarchitektur in der ImageNet Large Scale Visual Recognition Challenge [RDS+15] von Krizhevsky et al. [KSH12] noch aus acht Schichten mit lernbaren Parametern bestand, folgte 2014 das VGGNet von Simonyan et al. [SZ14] mit 19 Schichten bis hin zum ResNet-152 [HZRS16] von He et al. Die Verwendung von *Residual Learning* ermöglicht das Training des ResNets aus 152 Schichten. Eine Auswahl von Netzarchitekturen wird in Kapitel 3.1 vorgestellt.

2.2.1 *Training*

Das Training von modernen Neuronalen Netzen hat das Ziel, eine möglichst geringe Fehlerrate für eine bestimmte Aufgabe zu erreichen. Dabei handelt es sich um ein Optimierungsproblem mit Millionen von Parametern. Durch ihre große Anzahl ist eine analytische Festlegung von optimalen Parametern oder gar das Ausprobieren aller möglichen Parameterkonfigurationen im *Brute-Force*-Ansatz unmöglich. Aus diesem Grund wird für das Trainingsverfahren der Neuronalen Netze der sogenannte Gradientenabstieg eingesetzt. Dieses iterative Verfahren minimiert eine Kostenfunktion, was auch die Leistungsfähigkeit des Netzes optimieren soll [GBC16, S. 80f].

2.2.2 *Gradientenabstieg*

Die Kostenfunktion reduziert die tatsächliche und die gewünschte Ausgabe des Neuronalen Netzes auf einen einzigen numerischen Wert, der im Falle des Gradientenabstiegs als Kosten bzw. Abweichung der tatsächlichen Ausgabe $\hat{y} = f_w(x)$ von der gewünschten Ausgabe y des Netzes interpretierbar ist. Das Gradientenabstiegsverfahren minimiert eine gegebene Kostenfunktion C, indem es die Netzwerkparameter w iterativ anhand der Richtung des negativen Gradienten der Kostenfunktion anpasst

$$w' = w - \eta \frac{\partial C(\hat{y}, y)}{\partial w} \tag{2.2.2}$$

und so in jedem Schritt die Netzwerkausgabe dem gewünschten Ergebnis annähert. Auf diese Weise wird ein lokales Minimum gefunden, das im Normalfall nicht dem globalen Minimum der Kostenfunktion entspricht [GBC16, S. 82]. Daher hat der Startpunkt (die initialen Netzparameter w) einen Einfluss auf das gefundene lokale Optimum. Die korrekte Wahl der Schrittweite η ist essenziell für die Effizienz und Konvergenz des Lernprozesses und wird als Lernrate bezeichnet. Wird die Lernrate zu klein gewählt, verläuft der Trainingsprozess entsprechend langsam und der Lernprozess kann in ein schlechteres lokales Minimum der Kostenfunktion konvergieren. Ist die Lernrate zu hoch, besteht die Gefahr, dass der Lernprozess durch die großen Anpassungen der Gewichte instabil verläuft und die Gradienten oszillieren [Roj96, S. 187ff]. Dies wird in Abbildung 2.2.2 visualisiert.

Es existiert eine Auswahl verschiedener Verfahren, nach denen die Parameter initial festgelegt werden können. In den meisten Fällen findet eine zufallsbasierte Parameterinitialisierung wie die Glorot Initialisierung [GB10] statt. Bei der Glorot Initialisierung werden die Parameter aus einer Normalverteilung mit Mittelwert 0 und Standardabweichung $\frac{1}{\sqrt{fan\_in}}$ gezogen, wobei fan_in der Anzahl von eingehenden Verbindung in das Neuron entspricht. Eine häufig verwendete Kostenfunktionen für Regressionsprobleme ist z. B. der mittlere quadratische Fehler (*Mean Squared Error*, MSE)

$$C_{MSE}(\hat{y}, y) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2. \tag{2.2.3}$$

Für die Optimierung wurden verschiedene Formen des Gradientenabstiegs entwickelt. Während der stochastische Gradientenabstieg (*Stochastic Gradient Descent*, SGD) ein Update der Gewichte nach Betrachtung eines jeden Trainingsbeispiels durchführt (siehe Formel 2.2.2), mittelt der „gewöhnliche“ Gradientenabstieg (*Gradient Descent*, GD) die Gewichtsänderungen über alle Trainingsbeispiele. Der Vorteil des SGD

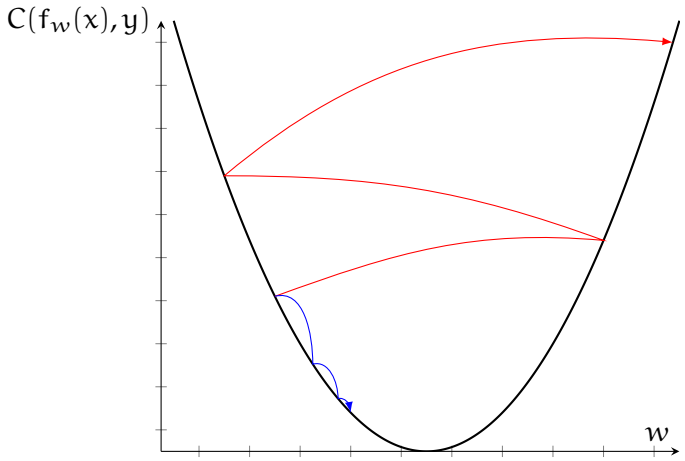


Abbildung 2.2.2: Visualisierung des Trainingsprozesses mit unpassender Lernrate. Die Darstellung zeigt den Wert einer Kostenfunktion C in Abhängigkeit eines bestimmten Gewichtes w vom Netzwerk f. Ist die Lernrate zu groß, kann w in zu großen Schritten angepasst werden und das Training divergiert. Ist sie zu klein, verläuft der Trainingsprozess langsam und das lokale Optimum wird gegebenenfalls nicht erreicht.

Verfahrens liegt darin, dass die Gewichtsaktualisierungen erheblich häufiger durchgeführt werden und dazwischen lediglich die Ausgabe für ein Trainingsbeispiel berechnet werden muss. Dies beschleunigt den Lernprozess gegenüber dem GD, hat aber einen negativen Einfluss auf die Konsistenz der berechneten Gewichtsänderungen. Daher wird heute oftmals Minibatch-basierter Gradientenabstieg (*Minibatch Gradient Descent*) eingesetzt, der die Gewichtsänderungen über eine kleine Anzahl von Trainingsbeispielen berechnet, die als Minibatch bezeichnet wird [GBC16, S. 149]. Die Größe des Minibatches hat außerdem einen entscheidenden Einfluss auf die Trainingseffizienz, da die Trainingsbeispiele eines Minibatches von spezieller Hardware wie etwa Grafikkarten oder Tensor Processing Units[1] in gewissen Grenzen Parallelisiert berechnet werden können, um einen Vorteil in der Trainingsgeschwindigkeit zu erzielen.

Es existieren zuzüglich zum Standard SGD Verfahren verschiedene Alternativen der Update-Regel für die Gewichte. Beim *Stochastic Gradient Descent with Momentum* wird versucht, dem Oszillieren des Gradienten durch Einbezug des gewichteten gleitenden

[1] siehe https://developer.nvidia.com/cudnn und https://cloud.google.com/tpu/

Durchschnitts vergangener Gradienten entgegenzuwirken. Durch die Wahl von γ kann in der Update-Regel

$$v_t = \gamma v_{t-1} + \eta \frac{\partial C(\hat{y}, y)}{\partial w}$$
$$w' = w - v_t$$

der Einfluss vom Momentum gesteuert werden. Verfahren wie AdaGrad [DHS11] oder dessen Erweiterungen RMSProp [TH12] und Adam [KB14] ergänzen das SGD Verfahren um an die einzelnen Parameter angepassten Lernraten. Die Lernrate für Parameter mit großen Gradienten wird im Vergleich zu Parametern mit kleinen Gradienten erheblich verringert. Dies führt zu einem schnelleren Fortschritt des Lernprozesses in Dimensionen mit kleinem absoluten Gradienten. Gleichzeitig kann das Oszillieren großer Gradienten durch die Verringerung ihrer effektiven Lernrate vermieden werden. Während AdaGrad eine Summe der Quadrate über alle vergangenen Werte der Gradienten betrachtet, nutzt RMSProp einen gewichteten gleitenden Durchschnitt, damit Gradienten aus der fernen Vergangenheit einen wesentlich geringeren Einfluss auf die aktuelle Lernrate haben. Adam (*Adaptive Moments*) verwendet gegenüber RMSProp einen gleitenden Durchschnitt vergangener Gradienten für die Lernrate und für das Momentum eines jeden Parameters [GBC16, S. 305f].

2.2.3 *Error Backpropagation*

Eine Grundvoraussetzung für die Verwendung des Gradientenabstiegverfahrens zum Training Neuronaler Netze, ist eine Möglichkeit für die effiziente Berechnung der aktualisierten Netzwerkparameter. Dafür wird der Gradient der Fehlerfunktion bezogen auf die Gewichte benötigt, der in Neuronalen Netzen mit dem *Error Backpropagation* Verfahren bestimmt wird. Die folgende Erläuterung orientiert sich an der Darstellung von Rumelhart et al. [RHW88].

Die Ableitung $\frac{\partial C}{\partial y_j}$ der Kostenfunktion C zu den Ausgaben y_i der letzten Schicht ist lediglich abhängig von der verwendeten Kostenfunktion. Berechnet ein Neuron j der Ausgabeschicht mit Neuronenverbindungen $i \to j$ die Funktion

$$y_j = \Theta(a_j) \qquad \text{mit } a_j = \sum_i y_i w_{ij}, \tag{2.2.4}$$

so kann die Ableitung von C nach a_j durch die Anwendung der Kettenregel

$$\frac{\partial C}{\partial a_j} = \frac{\partial C}{\partial y_j} \cdot \frac{dy_j}{da_j} \tag{2.2.5}$$

bestimmt werden. $\frac{dy_j}{da_j}$ entspricht dabei der Ableitung der Aktivierungsfunktion Θ. Somit kann berechnet werden, welchen Einfluss eine Veränderung der Eingabe x_j auf den Wert der Fehlerfunktion hat. Für die Aktualisierung der Gewichte w_{ij} (siehe Kapitel 2.2.2) wird jedoch die Veränderung der Kostenfunktion bezogen auf die Gewichte betrachtet. Diese ist gegeben durch

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial a_j} \cdot \frac{\partial a_j}{\partial w_{ij}} = \frac{\partial C}{\partial y_j} \cdot \frac{dy_j}{da_j} \cdot y_i. \tag{2.2.6}$$

Alle ausgehenden Verbindungen von i führen zusammen zu

$$\frac{\partial C}{\partial y_i} = \sum_j \frac{\partial C}{\partial a_j} \cdot \frac{\partial a_j}{\partial y_i} = \sum_j \frac{\partial C}{\partial y_j} \cdot \frac{dy_j}{da_j} \cdot w_{ij}. \tag{2.2.7}$$

Damit ist die Berechnung von $\frac{\partial C}{\partial y_i}$ für alle Neuronen i der vorletzten Schicht bekannt, wenn $\frac{\partial C}{\partial y_j}$ für die Neuronen j der letzten Schicht gegeben ist. Das gleiche Vorgehen kann für Neuronen l mit Verbindungen $l \to k$ vorhergehender Schichten durchgeführt werden, sodass $\frac{\partial C}{\partial y_k}$ und damit auch die entsprechenden Gradienten $\frac{\partial C}{\partial w_{lk}}$ sukzessive berechnet werden können.

Durch die wiederholte Anwendung der Kettenregel, die den Fehler jeweils eine Schicht weiter von der Ausgabe- zur Eingabeschicht zurückführt, wird dieses Vorgehen als Error Backpropagation bezeichnet. In vielen modernen Deep Learning Frameworks, wie z. B. PyTorch [PGC+17] und TensorFlow [AAB+15], können die erforderlichen Gradienten automatisiert berechnet werden.

Komplikationen entstehen durch das bis zu n-fache Produkt der Gradienten innerhalb der einzelnen Schichten, wenn das Netz insgesamt aus n Schichten besteht. Wird als Aktivierungsfunktion der Neuronen beispielsweise die Sigmoidfunktion (siehe 2.2.1) gewählt, liegt der Gradient $\frac{dy_j}{da_j}$ im Intervall $(0, \frac{1}{4})$ und das Produkt wird schnell extrem klein. Dies verlangsamt den Lernprozess und führt dazu, dass die sich ergebenden Werte so klein werden können, dass sie durch numerische Ungenauigkeiten, die sich bei der Verarbeitung im Computer ergeben, nicht mehr von 0 unterschieden werden können. Damit findet in diesen Neuronen kein Lernprozess mehr statt. Dies wird als *Vanishing Gradient Effect* bezeichnet, wobei das gegensätzliche Problem der *Exploding Gradients* für große Gradienten gerade bei der Anwendung von rekurrenten Netzen (siehe Kapitel 2.5) ebenfalls existiert [PMB13].

Heute werden neben dem Pre-Training des Netzes [PMB13] und der Anwendung von speziellen Schichten (wie z. B. LSTMs, siehe Kapitel 2.5.1) verschiedene Techniken verwendet, um das Vanishing und Exploding Gradient Problem zu verhindern, den Trainingsprozess zu beschleunigen und die Leistung des Netzes zu optimieren. Eine Auswahl dieser Methoden wird in Kapitel 2.4 vorgestellt.

2.3 FALTUNGSNETZE

Ein wichtiger Baustein für den Erfolg Neuronaler Netze im Bereich der Bildverarbeitung sind die Faltungsnetze. Bei der Verarbeitung eines Farbbildes, beispielsweise der Größe 100×100, sind die Eingabedaten so hoch-dimensional, dass ein einzelnes Neuron der ersten Schicht eines MLPs bereits 30 000 Gewichte besitzen würde. Der Speicherverbrauch und Rechenleistungsbedarf wird so hoch, dass auch moderne Computerhardware nicht ausreicht, um ein solches MLP zu trainieren. Zudem führt die hohe Modellkapazität selbst bei großen Datensätzen zur Überanpassung (siehe 2.4.1). Aus diesen Gründen setzen viele moderne Ansätze im Bereich des maschinellen Lernens und im Speziellen der Bildverarbeitung auf Faltungsnetze (*Convolutional Neural Networks*, CNNs), die im Kern aus sogenannten Faltungs-, Pooling-, voll-verbundenen und Softmax-Schichten bestehen. Diese Schicht-Typen sollen in den folgenden Abschnitten erläutert werden.

2.3.1 *Faltungsschichten*

Zentral für die Merkmalsextraktion innerhalb von Faltungsnetzen sind die Faltungsschichten, die alle Eingaben mit einer Maske falten, die sich aus den gelernten Parametern ergibt. Der Teil der Eingabe, der für die Berechnung eines bestimmten Elements der Ausgabe relevant ist, wird als Rezeptives Feld bezeichnet. Essenziell ist, dass die Gewichte einer Faltungsschicht von den Neuronen geteilt werden, sodass sich im Vergleich zu voll-verbundenen Schichten eine verringerte Parameteranzahl ergibt und die berechneten Merkmale eine Translationsinvarianz besitzen [LBBH98]. Dies ermöglicht ein effizienteres Training, da jeder Filter, der sich aus den Gewichten ergibt, nur einmal erlernt werden muss. Dieser wird daraufhin auf die gesamte Eingabe angewendet, weshalb sich Faltungsschichten besonders in der Bildverarbeitung etabliert haben. Oftmals werden in einer Faltungsschicht mehrere Filter angewendet und die sich ergebenden Ausgaben, die als Merkmalskarten bezeichnet werden, ähnlich zu Farbkanälen in Bildern hintereinander angeordnet. Dies wird in Abbildung 2.3.1 dargestellt.

Jedes Neuron an der Stelle (u, v) der Merkmalskarte c_{out} wendet die Funktion

$$a_{c_{out},u,v} = b_{c_{out}} + \sum_{s_u=-\lceil \frac{k_u}{2}-1 \rceil}^{\lfloor \frac{k_u}{2} \rfloor} \sum_{s_v=-\lceil \frac{k_v}{2}-1 \rceil}^{\lfloor \frac{k_v}{2} \rfloor} \sum_{c_{in}=1}^{C_{in}} w_{c_{out},c_{in},s_u,s_v} \cdot x_{c_{in},u+s_u,v+s_v} \quad (2.3.1)$$

an, wobei $k_u \times k_v$ die Größe des Rezeptiven Feldes, C_{in} die Anzahl der Eingabekanäle, C_{out} die Anzahl der Ausgabekanäle, $w_{c_{out},c_{in},s_u,s_v}$ die Gewichte, $b_{c_{out}}$ der Bias und

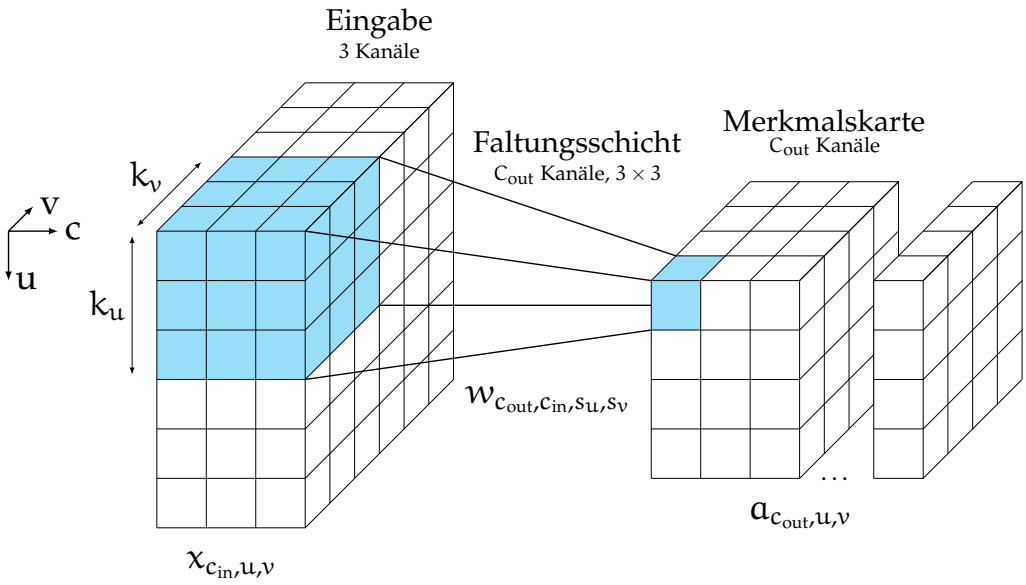


Abbildung 2.3.1: Visualisierung einer Faltungsschicht nach [Sud18]. Um ein Element der ausgegebenen Merkmalskarte zu berechnen, wird die gewichtete Summe über das Rezeptive Feld in der Eingabe und die Parameter $\boldsymbol{w}$ gebildet.

$x_{c_{in},u+s_u,v+s_v}$ die Eingabe ist. Für eine Kernelgröße von 3×3 und einem einzelnen Ausgabekanal ergibt dies

$$a_{u,v} = b + \sum_{s_u=-1}^{1} \sum_{s_v=-1}^{1} \sum_{c_{in}=1}^{C_{in}} w_{c_{in},s_u,s_v} \cdot x_{c_{in},u+s_u,v+s_v}. \tag{2.3.2}$$

Durch diese Operation ist die ausgegebene Merkmalskarte sowohl in u- als auch v-Richtung um zwei Elemente kleiner als die Eingabe. Soll dies vermieden werden, kann die Eingabe vorher in jeder Richtung um die entsprechende Anzahl Zeilen bzw. Spalten erweitert werden, was als *Padding* bezeichnet wird. Die Schrittweite (*Stride*) gibt den Abstand zwischen den Mittelpunkten der Rezeptiven Felder benachbarter Neuronen an. Hierdurch kann eine Überlappung oder sogar eine Lücke zwischen den Rezeptiven Feldern erzeugt werden. Es existieren weitere Parameter wie die Dilation, mit denen Einfluss auf die Gestalt des rezeptiven Feldes genommen werden kann.

2.3.2 *Pooling-Schichten*

Pooling-Schichten fassen die Ausgaben zusammenhängender Bereiche von Neuronen zusammen, um die Größe der zu verarbeitenden Daten für die nachfolgenden Schichten zu reduzieren und eine Translationsinvarianz zu erzeugen [SMB10]. Die Reduktion der Datenmenge wirkt außerdem der Überanpassung des Netzes entgegen. Damit werden tiefere Netzarchitekturen mit einer größeren Anzahl von hintereinander geschalteten Schichten begünstigt.

Der Kernel gibt die Größe des zusammengefassten Eingabebereichs an und hat oft die Größe 2×2, sodass die Größe der eingegebenen Merkmalskarte auf ein Viertel reduziert wird. Ähnlich zu den Faltungsschichten existieren weitere Hyperparameter zur Beeinflussung des zu reduzierenden Eingabebereichs wie Padding oder Stride. Eine oft verwendete Pooling-Variante ist das Maximum Pooling, bei dem sich die Ausgabe aus dem Maximum des Eingabebereichs ergibt. Es existieren jedoch Alternativen wie z. B. das Average Pooling, bei dem die Ausgabe dem Mittelwert der Eingabemerkmale entspricht und das Adaptive Pooling[2], bei dem keine Kernelgröße sondern die gewünschte Größe der Ausgabe vorgegeben wird. Letzteres ist beispielsweise hilfreich, wenn Eingabedaten unterschiedlicher Größen verarbeitet werden sollen, nachfolgende Schichten, wie z. B. Fully-Connected Layer, aber eine feste Größe der Eingabe erwarten.

2.3.3 *Voll-verbundene Schichten*

Trotz der vermehrten Verwendung spezialisierter Schichttypen wie der Faltungsschicht finden auch voll-verbundene Schichten (*Fully-Connected Layer*) Einsatz in aktuellen Architekturen von Faltungsnetzen (siehe [KSH12], [SZ14]). Hier werden sie in der Regel als Klassifikatoren eingesetzt, die ein Muster anhand seiner Merkmale einer Klasse zuordnen. Außerdem können sie die Informationen vorheriger Schichten in das benötigte Ausgabeformat bringen. Werden als Ausgabe beispielsweise Pseudowahrscheinlichkeiten für eine Reihe von Klassen innerhalb einer Klassifikationsaufgabe benötigt, wird dies oft durch eine voll-verbundene Schicht in Kombination mit einer nicht-linearen Aktivierungsschicht wie Softmax (siehe folgender Abschnitt) durchgeführt. Weil voll-verbundene Schichten meist in den hinteren Schichten der Architektur nach der Reduktion der Merkmale (z. B. durch Pooling-Schichten) eingesetzt werden, bleibt ihre Parameteranzahl akzeptabel.

[2] siehe https://pytorch.org/docs/stable/nn.html#adaptivemaxpool2d

2.3.4 *Softmax-Schichten*

Softmax-Schichten können als finale Schicht eines Neuronalen Netzes verwendet werden, um die Pseudowahrscheinlichkeit einer Klassenzugehörigkeit aus K Klassen zu berechnen. Hierfür werden die Ausgabewerte der vorherigen (meist voll-verbundenen) Schicht mithilfe der Formel

$$\Theta_{Softmax}(\mathbf{a})_j = \frac{\exp(a_j)}{\sum_{k=1}^{K} \exp(a_k)} \tag{2.3.3}$$

in den Wertebereich $[0, 1]$ gebracht, sodass die Summe aller Werte $\sum_{k=1}^{K} \sigma(a)_k = 1$ ergibt. Der Name Softmax ist mit dem Verhalten der Funktion zu erklären. Ist $a_i \gg a_j$ für alle $j \neq i$ so gilt $\Theta(a)_i \approx 1$ und $\Theta(a)_j \approx 0$ [Biso6, S. 198]. Auf diese Weise kann die Ausgabe des Netzes $\Theta(a)_k$ als Pseudowahrscheinlichkeit für die Zugehörigkeit der Eingabe zur Klasse k betrachtet werden.

2.4 TRAININGSOPTIMIERUNG

Beim Training von Neuronalen Netzen mit Millionen von Parametern mithilfe des Gradientenabstieg-Verfahrens ergeben sich einige technische Hürden. Überanpassung, der Vanishing Gradient Effekt und der Berechnungsaufwand stellen in jeder Anwendung von tiefen Neuronalen Netzen zentrale Anforderungen an den Trainingsprozess und die verwendete Netzarchitektur. Für deren Bewältigung wurde eine Vielzahl von Techniken entwickelt, von denen im Folgenden einige Beispiele erläutert werden.

2.4.1 *Überanpassung*

Durch die große Anzahl von lernbaren Parametern sind Neuronale Netze in der Regel sehr gut dazu in der Lage den Fehler auf dem Trainingsdatensatz innerhalb der Optimierung zu minimieren. Problematisch wird dies, wenn die Generalisierungsleistung des Netzes durch den Optimierungsprozess beeinträchtigt wird und damit der Fehler auf neuen Eingabedaten durch das Training wächst. Dieses Verhalten wird auch als Überanpassung (*Overfitting*) bezeichnet. Überanpassung steht im direkten Verhältnis zur Modellkapazität, der Möglichkeit eines Modells sich an eine Vielfalt von Funktionen anpassen zu können. Eine hohe Modellkomplexität hat, wenn keine ausreichenden Gegenmaßnahmen eingesetzt werden, zur Folge, dass das Netz bestimmte Eigenschaften des Trainingsdatensatzes auswendig lernt. Dies führt zu einer verminderten Leistungsfähigkeit, wenn diese Eigenschaften auf neuen Eingabedaten

wie z. B. einem Testdatensatz nicht vorhanden sind [GBC16, S. 110]. Innerhalb der Forschungsgemeinschaft um Neuronale Netze wurden verschiedene Techniken entwickelt, um Überanpassung entgegenzuwirken, die auch als Regularisierung bezeichnet werden. Eine Regularisierungsmöglichkeit besteht in der Anwendung eines „Strafterms" innerhalb der Kostenfunktion. Hier kann ein gewisser Anteil der Summe über die absoluten (L_1-Regularisierung) oder quadrierten (L_2-Regularisierung) Modellgewichte auf die eigentliche Kostenfunktion addiert und als neue Kostenfunktion verwendet werden. Während L_2-Regularisierung lediglich die Größe der gelernten Gewichte vermindert, sorgt die L_1-Regularisierung dafür, dass einige Parameter null werden [GBC16, S. 227ff]. Dünn besetzte Gewichtsmatrizen sollen nach [CIR12] zu besseren diskriminativen Merkmalen führen.

EARLY STOPPING Das vorzeitige Abbrechen des Trainingsvorgangs, bevor der Trainingsfehler vollständig minimiert ist, wird als *Early Stopping* bezeichnet. Es handelt sich um eine Maßnahme gegen Überanpassung, deren Intention in Abbildung 2.4.1 erkennbar ist. Über das vorzeitige Beenden des Optimierungsprozesses soll der ansteigende Teil der Validierungsfehler-Kurve vermieden werden [Pre12]. Eine Schwierigkeit besteht allerdings in der korrekten Festlegung des Stopp-Kriteriums. Oft wird nach jeder Trainingsepoche eine Evaluation auf einem Validierungsdatensatz durchgeführt, um den korrekten Early-Stopping Zeitpunkt zu finden. Ergibt sich nach einer festgelegten Anzahl von Epochen keine weitere Verbesserung, wird der Trainingsvorgang abgebrochen und das Modell mit dem bisher besten Ergebnis auf dem Validierungsdatensatz übernommen.

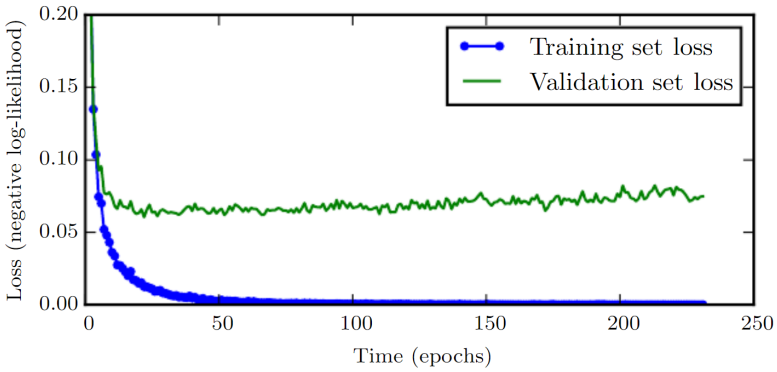
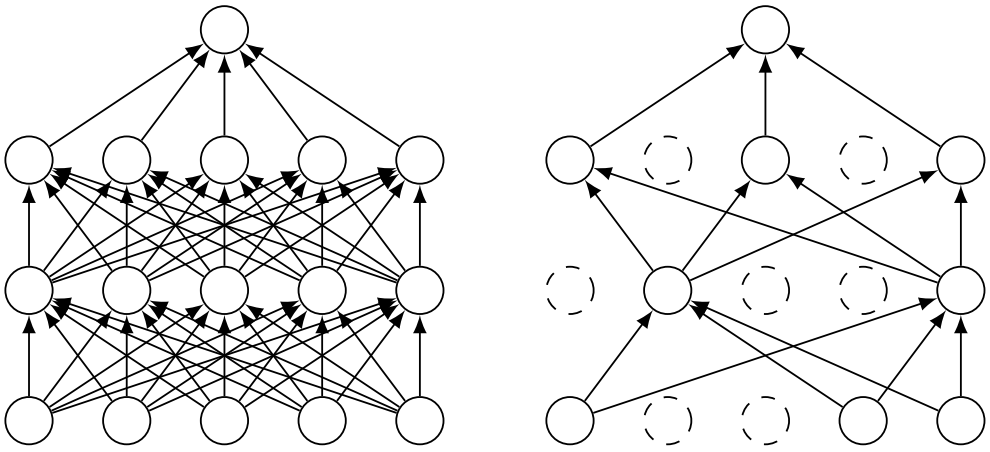


Abbildung 2.4.1: Gegenüberstellung der Trainings- und Validierungs-Kosten je Epoche aus [GBC16]. Die Kosten auf dem Testdatensatz steigen etwa ab der 30. Trainingsepoche, bei kontinuierlich fallenden Kosten auf dem Trainingsdatensatz.

DROPOUT Eine weitere Maßnahme um Überanpassung in Neuronalen Netzen zu verhindern ist *Dropout* [HSK+12], bei dem für jeden Durchlauf ein Anteil p der Aktivierungen von zufällig gewählten Neuronen auf null gesetzt und das Neuron damit deaktiviert wird. Dies ist in Abbildung 2.4.2 visualisiert. Die übrigen Neuronen werden zum Ausgleich mit $\frac{1}{1-p}$ skaliert, damit die Gesamtaktivierung gegenüber dem Netz ohne Dropout unverändert bleibt. Durch Dropout wird verhindert, dass die Neuronen der folgenden Schicht im Trainingsprozess von einem kleinen Teil ihrer Eingabe abhängig werden. Dieser Ko-Adaption der Neuronen wird durch die zufallsbasierte Deaktivierung der Eingaben entgegengewirkt. In Faltungsschichten kann es trotz Dropout durch die hohe Korrelation benachbarter Neuronen im Rezeptiven Feld zur Ko-Adaption kommen. Um das zu verhinden kann Spatial Dropout [TGJ+15] eingesetzt werden, dass einen ganzen Filter statt einzelner Aktivierungen deaktiviert, sodass alle Aktivierungen einer Merkmalskarte gemeinsam aus der Eingabe der Folgeschicht entfernt werden.

Dropout kann als Verwendung eines Ensembles von 2^n Netzen verstanden werden, die sich in großem Maß Parameter teilen [HSK+12]. Während die Auswertung eines so großen Ensembles von Neuronalen Netzen durch die erforderlichen Berechnungen zu aufwändig wäre, kann Dropout in der Evaluationsphase des Netzes einfach durch die Identitätsfunktion ersetzt werden. Ein Nachteil von Dropout besteht darin, dass im deaktivierten Teil der Neuronen in dem Trainingsschritt keine Gewichtsaktualisierung durchgeführt wird. Dies kann den Trainingsprozess insgesamt verlangsamen.



(a) (b)

Abbildung 2.4.2: (a) Ein Multi-Layer Perceptron ohne Dropout und (b) mit Dropout. Abbildungen nach [SHK+14].

AUGMENTIERUNG Überanpassung kann auch durch eine große Menge von Trainingsdaten verhindert werden. Da diese jedoch normalerweise mit den verfügbaren Datensätzen eine feste Größe besitzen, die nur durch erheblichen Mehraufwand aufgrund der erforderlichen manuellen Annotation erweitert werden kann, wird versucht die Menge der Trainingsdaten künstlich zu erhöhen. Bei der Verwendung von Neuronalen Netzen ist es innerhalb vieler Anwendungsfälle möglich die Eingabedaten und ihre dazugehörigen Annotationen zu transformieren. Dieser Vorgang wird als Augmentierung bezeichnet und ist eine zentrale Methodik, die Verwendung von tiefen Neuronalen Netzen auch in Anwendungsfällen mit verhältnismäßig kleinen Datensätzen zu ermöglichen. Innerhalb der Klassifikation von Bildern können beispielsweise Bildinhalte gespiegelt, gedreht oder gedehnt, die Helligkeit und weitere Bildparameter leicht verändert, Rauschen hinzugefügt oder Bildausschnitte betrachtet werden (vgl. [KSH12, CMS12, SSP03]). Diese Operationen verändern die Bedeutung des Bildinhalts nicht, sodass die ursprüngliche Annotation weiterhin gültig ist. Im Falle von Objekterkennung müssen bei einigen der genannten Operationen aber die Informationen zu den Objekt-Grenzen der Annotation mit-transformiert werden. Diese Berechnungen sind jedoch leicht durchzuführen. Weitere Augmentierungsoperationen, speziell für die Anwendung in der Handschrifterkennung, werden in Kapitel 3.6 vorgestellt.

Augmentierung kann auch zur Leistungssteigerung innerhalb der Evaluation eingesetzt werden, indem alle Eingabedaten mehrfach augmentiert, durch das Netz verarbeitet und die Netzausgaben gemittelt werden. Dies wird als Test-Side Augmentation bezeichnet [WSD+17].

2.4.2 *Rectified Linear Units*

Als Aktivierungsfunktion werden heute oft *Rectified Linear Units* (ReLUs) [NH10]

$$\Theta_{\mathrm{ReLU}}(a) = \begin{cases} a, & a \geqslant 0 \\ 0, & a < 0 \end{cases} \tag{2.4.1}$$

eingesetzt. Im Vergleich zu anderen Aktivierungsfunktionen, wie der Sigmoidfunktion, besitzen die ReLUs einen konstanten Gradienten von null im negativen Teil und von eins im positiven Teil des Definitionsbereichs. Im Vergleich dazu ist der Gradient der Sigmoid Aktivierungsfunktion nahe null, wenn $|a|$ groß ist. Bei einem mehrschichtigen Neuronalen Netz werden im Training die Gradienten mehrerer Schichten miteinander multipliziert, was im Falle der Sigmoidfunktion eine mehrfache Multiplikation mit sehr kleinen Zahlen bedeutet. Die ReLUs umgehen durch die Multiplikation von Einsen für

Pfade aktivierter Neuronen numerische Probleme bei der Fließkommarepräsentation kleiner Zahlen. Sie wirken damit dem Vanishing Gradient Effekt entgegen [MHN13], sodass auch tiefe Netze erfolgreich traininert werden können.

Von den Rectified Linear Units existieren verschiedene Varianten, wie die *Leaky ReLUs* [MHN13] oder die *Exponential Linear Units* (ELUs) [CUH15]

$$\Theta_{\text{LeakyReLU}}(a) = \begin{cases} a, & \text{falls } a \geqslant 0 \\ 0.01a, & \text{sonst} \end{cases} \tag{2.4.2}$$

$$\Theta_{\text{ELU}}(a) = \begin{cases} a, & \text{falls } a \geqslant 0 \\ \alpha(\exp(a) - 1), & \text{sonst} \end{cases}. \tag{2.4.3}$$

Während bei der Verwendung von ReLUs als Aktivierungsfunktion in den nicht aktivierten Neuronen durch den Gradienten von null kein Lernprozess stattfindet, ersetzen die Leaky ReLUs (siehe Gleichung 2.4.2) den Null-Gradienten der ReLUs durch einen kleinen Wert (z. B. 0.01) Die ELUs (siehe Gleichung 2.4.3) haben ebenso wie die LeakyReLUs den Vorteil, dass negative Werte für die Aktivierung möglich sind und damit die mittlere Aktivierung im Vergleich zu den ReLUs näher an null liegt. Ist die mittlere Aktivierung nicht null wirkt dies wie ein Bias, den ein verbundenes Neuron im darauffolgenden Layer ausgleichen muss. Deshalb soll die Verwendung von LeakyReLUs oder ELUs den Trainingsprozess beschleunigen [CUH15]. Darüber hinaus sollen die ELUs im Vergleich zu den anderen ReLU-Varianten robuster gegenüber Rauschen in den Aktivierungen sein, weil der Gradient der ELU-Aktivierungsfunktion stetig ist. So können kleinste Veränderungen der Aktivierung keinen großen Einfluss auf den Gradienten haben [CUH15]. In Abbildung 2.4.3 werden die genannten Aktivierungsfunktionen dargestellt.

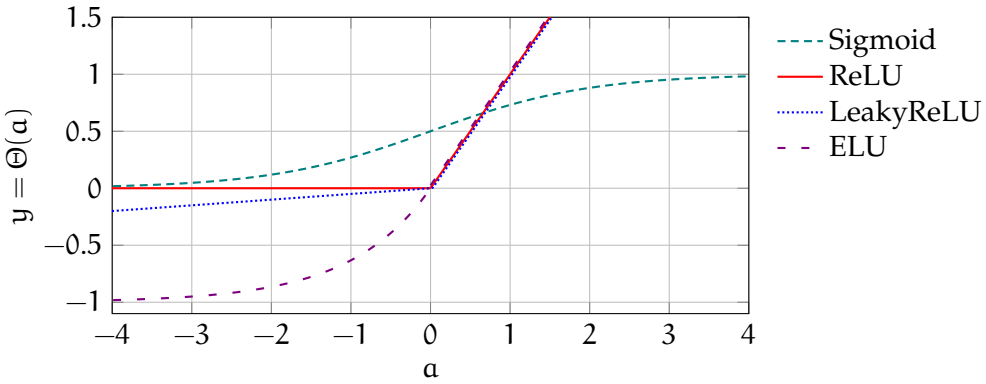


Abbildung 2.4.3: Visualisierung verschiedener Aktivierungsfunktionen.

2.4.3 *Batch Normalisierung*

Normalisierung von Eingabedaten ist eine lineare Transformation, sodass diese einen Mittelwert von null und Standardabweichung von eins haben. Dies ist eine gängige Methode für die Beschleunigung von Trainingsvorgängen [LBOM98]. Durch den Lernprozess und die sich dadurch anpassenden Gewichte verändert sich die Verteilung der Eingaben in den versteckten Schichten des Netzes ebenfalls, was als interne Kovariate Verschiebung bezeichnet wird. Batch Normalisierung [IS15] sorgt für eine stabilere Verteilung der Eingabedaten einer Schicht auch innerhalb des Netzes und ermöglicht somit ein von vorhergehenden Schichten unabhängigeres Training. Für die Batch Normalisierung wird der Mittelwert und die Varianz aller Skalare des Eingabe-Batches berechnet und diese damit normalisiert. Außerdem wird der Skalierungsparameter γ und der Verschiebungsparameter β für jedes Neuron eingeführt, die innerhalb des Trainingsvorgangs mit angepasst werden. Ohne die beiden Parameter wäre die Modellkapazität zu sehr eingeschränkt, da beispielsweise die Eingabe in eine Sigmoid Aktivierungsfunktion durch die Batch Normalisierung auf ihren linearen Teil beschränkt sein würde. Insgesamt berechnet die Batch Normalisierung ihre Ausgabe y_b durch die Funktion

$$\mu_B = \frac{1}{|B|}\sum_{b=1}^{|B|} x_b \qquad \sigma_B^2 = \frac{1}{|B|}\sum_{b=1}^{|B|}(x_b - \mu_B)^2 \qquad \hat{x}_b = \frac{x_b - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$y_b = \gamma \hat{x}_b + \beta \tag{2.4.4}$$

für einen Batch B der Größe $|B|$ und die Eingabe x. ϵ ist eine kleine Konstante, die lediglich der numerischen Stabilität dient. Innerhalb der Evaluierung werden die Mittelwerte und Varianzen nicht batchbasiert berechnet, sondern die während des Trainingsprozesses aufgezeichneten laufenden Mittelwerte μ_B und σ_B^2 verwendet.

Durch die größere Unabhängigkeit gegenüber den Ausgabeverteilungen der vorherigen Layer kann oft die Lernrate und damit die Geschwindigkeit des Trainingsprozesses erhöht werden. Die Möglichkeit der Reduzierung von Dropout beim Einsatz von Batch Normalisierung im Neuronalen Netz begünstigt diesen Effekt zusätzlich [IS15]. Die Ausgabe für ein bestimmtes Trainingsbeispiel ist bei der Verwendung von Batch Normalisierung nicht deterministisch, da μ_B und σ_B^2 von den Daten des gesamten Batches abhängen.

2.5 REKURRENTE NEURONALE NETZE

Für bestimmte Aufgaben, wie der Handschrifterkennung, ist es wertvoll, die aktuell zu analysierende Information in ihrem lokalen und globalen Kontext betrachten zu können. Daten solcher Prozesse liegen oft als Sequenz vor, sodass Verfahren benötigt werden, die solche Sequenzen unterschiedlicher Länge verarbeiten können. Einige Faltungsnetze sind zwar dazu in der Lage, Bilder variabler Größe zu verarbeiten und lokalen Kontext innerhalb des Rezeptiven Feldes in die Berechnungen einfließen zu lassen, jedoch fehlt ihnen die sequenzbasierte Spezialisierung [GBC16, S. 367]. Für die Betrachtung des globalen Kontextes, also die auf die Gesamtheit der Eingabedaten bezogenen Informationen, bedarf es innerhalb der Netzarchitektur Kantenverbindungen zwischen den Neuronen einer Schicht bzw. zu einer vorhergehenden Schicht. Diese Neuronenverbindungen sind in Feed-Forward Netzen ausgeschlossen und ergeben deshalb eine neue Kategorie mit der Bezeichnung rekurrente Neuronale Netze (*Recurrent Neural Networks*, RNNs). Effektiv bekommt das Neuronale Netz mit den rekurrenten Verbindungen eine Art Gedächtnis über vorherige Elemente der Sequenz, das die aktuelle Ausgabe beeinflussen kann [Grao8]. Eine einfache rekurrente Schicht ist in Abbildung 2.5.1 dargestellt.

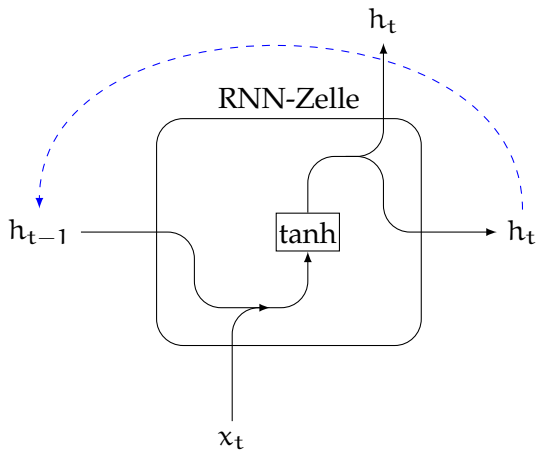


Abbildung 2.5.1: Darstellung einer Zelle eines rekurrenten Neuronalen Netzes nach [Chr15b]. Der gestrichelte Pfeil entspricht der rekurrenten Schleife mit einer Verzögerung von einem Zeitschritt.

Das Hauptproblem der naiven Implementierung eines rekurrenten Netzes besteht in der Empfindlichkeit ihres Trainingsprozesses. Die Entfaltung der rekurrenten Struktur entlang der zeitlichen Dimension und das Training per Backpropagation ist ein geeigneter Algorithmus für die Aktualisierung der Parameter. Dieser wird als *Backpropagation Through Time* (BPTT) bezeichnet und ist in Abbildung 2.5.2 dargestellt. Es bestehen mit dieser einfachen Architektur jedoch einige Probleme. Zum einen ist der Umfang des praktisch nutzbaren Kontextes sehr klein, da der Einfluss einer bestimmten Eingabe auf die Ausgabe mit der Sequenzlänge entweder laufend abnimmt oder exponentiell zunimmt [HS97]. Außerdem leiden diese Netze im Training unter einer gravierenden Vanishing bzw. Exploding Gradient-Problematik. Ein einfaches rekurrentes Neuronales Netz unter Auslassung der Aktivierungsfunktion Θ und der Eingaben $\mathbf{x}$ kann durch die Matrixmultiplikation $\mathbf{h}^{(t)} = \boldsymbol{W}^{\intercal}\mathbf{h}^{t-1}$ dargestellt werden. Die rekurrente Relation bei t Zeitschritten kann vereinfacht durch $\mathbf{h}^{(t)} = (\boldsymbol{W}^{t})^{\intercal}\mathbf{h}^{(0)}$ beschrieben werden. Hat $\boldsymbol{W}$ die Eigenwertzerlegung $\boldsymbol{W} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}_{\intercal}$, so ist die rekurrente Beziehung durch

$$\mathbf{h}^{t} = \boldsymbol{Q}^{\intercal}\boldsymbol{\Lambda}^{t}\boldsymbol{Q}\mathbf{h}^{(0)} \tag{2.5.1}$$

bestimmt. Die Eigenwerte werden mit t potenziert, sodass Eigenwerte Λ_i mit $|\Lambda_i| > 1$ explodieren und Eigenwerte mit $|\Lambda_i| < 1$ verschwinden. Dieser Effekt kann dazu führen, dass der Lernprozess langsam abläuft oder gar nicht erfolgreich abgeschlossen werden kann [GBC16, S. 396ff].

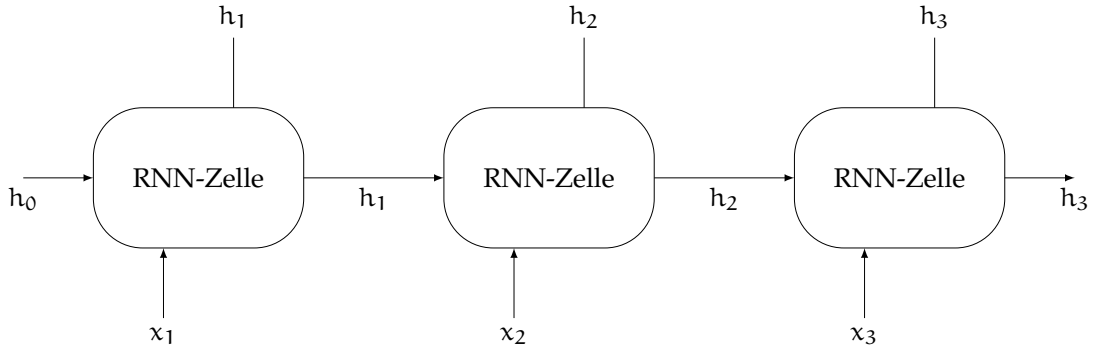


Abbildung 2.5.2: Entfaltetes rekurrentes Netz für die Backpropagation Through Time nach [Chr15b]. In jedem Zeitschritt t wird x_t eingegeben und h_t ausgegeben. Für die Berechnung der Ausgabe h_t wird die Ausgabe des vorherigen Zeitschritts h_{t-1} miteinbezogen.

Die *Long Short-Term Memory* und *Gated Recurrent Unit* Schichten sollen diesem Problem durch einen spezialisierten Aufbau der Zellen und ihrer Verbindungen entgegenwirken.

2.5.1 *Long Short-Term Memory*

Das Kern-Ziel der Long Short-Term Memory (LSTM) Schicht, ist die Erzeugung eines konstanten Fehlerrückflusses innerhalb der Backpropagation, also die Vermeidung eines Vanishing oder Exploding Gradients [HS97]. Erreicht wird dies durch die Verwendung eines internen Zustandes, der unverändert an den nächsten Zeitschritt weitergegeben und nur durch sogenannte Gates verändert wird. Die in dieser Arbeit verwendete Implementation[3] verwendet LSTMs mit Forget Gate [GSC00] und berechnet die Funktion

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \\
c_t &= f_t * c_{(t-1)} + i_t * g_t \\
h_t &= o_t * \tanh(c_t)
\end{aligned}
$$

wobei σ der Sigmoidfunktion, $*$ dem elementweisen Matrizen-Produkt, x_t den Eingaben und W den Gewichten entspricht. Eine Visualisierung ist in Abbildung 2.5.3 zu finden. Die Input-, Output- und Forget-Gates der Speicherzellen, aus denen die Schicht aufgebaut wird, steuern den Informationsfluss. Sie entscheiden, welcher Teil der Information in den internen Zustand aufgenommen, aus diesem ausgegeben oder vergessen wird. Jedes Gate besteht dabei im Kern aus jeweils einer Sigmoid-Schicht und einer elementweisen Multiplikation.

Im Falle des Forget-Gates f_t entscheiden die Ausgaben der vorherigen Speicherzelle $h_{(t-1)}$ und die Eingaben für die aktuelle Speicherzelle x_t über den Anteil der Information im internen Zustand, der übernommen wird. Sie werden konkateniert und durch eine Sigmoid-Schicht in den Wertebereich $[0, 1]$ transformiert und können so durch eine elementweise Multiplikation mit den Elementen des internen Zustands $c_{(t-1)}$ Einfluss auf die gespeicherte Information c_t nehmen. Die Funktionsweise des Input-Gates i_t ist dem Forget-Gate ähnlich. Eine Tangens hyperbolicus (tanh) Schicht bestimmt aus x_t und $h_{(t-1)}$ die hinzuzufügende Information g_t, die mit der gleichen

[3] siehe https://pytorch.org/docs/stable/nn.html#lstm

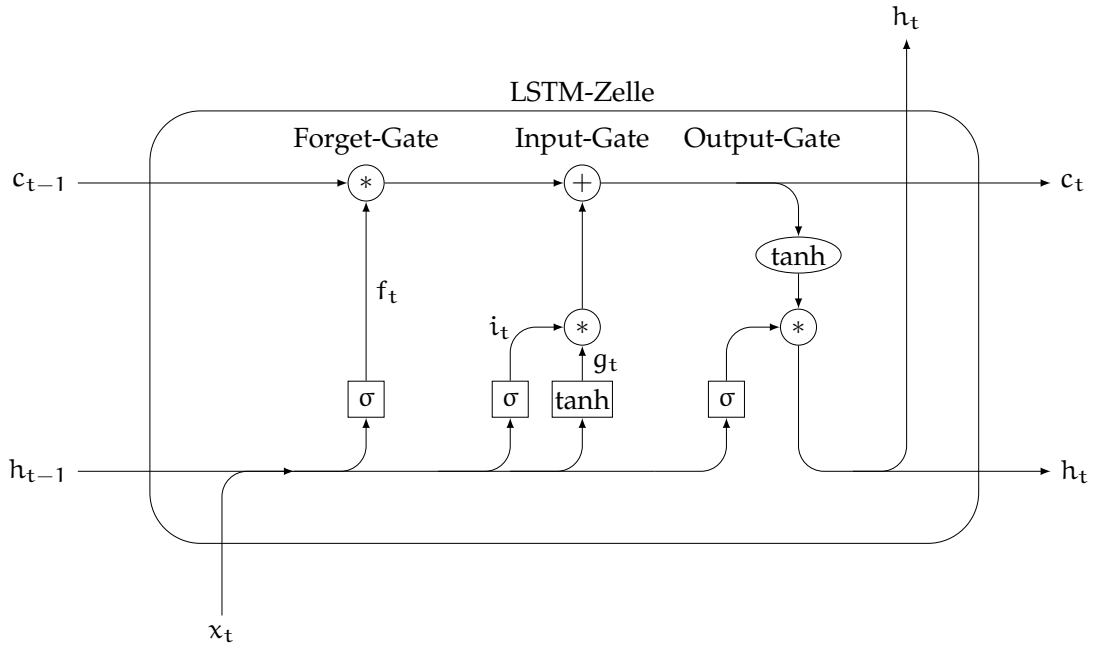


Abbildung 2.5.3: Visualisierung der Forget-, Input- und Output-Gates einer LSTM Zelle nach [Chr15b]. Die rechteckigen Knoten sind Schichten mit gelernten Parametern, die runden und elliptischen Knoten entsprechen elementweisen Operationen.

Kombination aus Sigmoid Schicht (siehe i_t) und elementweise Multiplikation gefiltert wird. Die so gefilterte Eingabe wird mithilfe einer elementweisen Addition dem internen Zellzustand hinzugefügt. Auch das Output-Gate o_t folgt dem gleichen Prinzip. Hier wird der Zellzustand c_t durch eine Tangens hyperbolicus Schicht in die Ausgabe h_t transformiert, die wieder durch die Sigmoid Schicht (siehe o_t) und elementweiser Multiplikation gefiltert wird. Die so gefilterte Ausgabe der aktuellen Zelle wird sowohl in die darüberliegende Schicht, als auch an die in zeitlicher Dimension nächste Zelle weitergegeben. Der interne Zustand c_0 und die Ausgabe h_0 für den ersten Zeitschritt werden meist konstant mit null initialisiert.

Während der Fehler innerhalb eines einfachen rekurrenten Netzes im exponentiellen Zusammenhang zur Größe der Eingabedaten in der zeitlichen Dimension steht, sollen diese Gates in den LSTMs für einen konstanten Fluss des Fehlers bei der Backpropagation sorgen. Der interne Zustand wird ohne eine multiplikative Veränderung durch Gewichte an die folgenden Zellen weitergegeben, sodass die darin enthaltenen

Werte nicht exponentiell wachsen oder schrumpfen. Hochreiter et al. nennen dies *constant error carrousel* [HS97].

Es existieren weitere Varianten von LSTM Architekturen, wie z. B. die ursprünglichen LSTMs ohne Forget Gate [HS97] oder Peephole-LSTMs [GS00]. Letztere führen gewichtete Verbindungen zwischen dem Zellzustand und den Gates der LSTM Zelle ein. Dies sorgt dafür, dass die Gates direkten Zugriff auf den Inhalt haben, den sie kontrollieren, was die Leistungsfähigkeit in bestimmten Anwendungsfällen steigern soll.

2.5.2 *Gated Recurrent Units*

Eine Alternative zur Long Short-Term Memory Schicht sind die sogenannten Gated Recurrent Units (GRUs) von Cho et al. [CMG+14]. Die interne Struktur ihrer Speicherzellen besteht, motiviert durch die LSTM Zellen, aus Gates. Im Gegensatz zu den LSTMs besitzen die GRUs jedoch nur zwei statt vier Gates, die als Reset- und Update-Gates bezeichnet werden. Gated Recurrent Units berechnen die Funktion

$$
\begin{aligned}
r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \\
z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \\
n_t &= \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{(t-1)} + b_{hn})) \\
h_t &= (1 - z_t) * n_t + z_t * h_{(t-1)}.
\end{aligned}
$$

und werden in Abbildung 2.5.4 dargestellt.

Das Update-Gate z_t steuert den Anteil des Informationsflusses von der vorherigen Ausgabe der Speicherzelle $h_{(t-1)}$ oder dem Zellzustand n_t in die neue Ausgabe der Speicherzelle h_t. Das Reset-Gate r_t entscheidet über den Einbezug der vorherigen Ausgabe $h_{(t-1)}$ für die Berechnung des neuen Zellzustands n_t. Ist das Reset-Gate nahe null, wird der Zellzustand dazu gezwungen die vorherige Ausgabe $h_{(t-1)}$ zu ignorieren, was es dem Zellzustand laut den Autoren ermöglicht irrelevante Information aus der Vergangenheit zu vergessen und eine kompaktere Repräsentation zu erlernen [CMG+14]. Die Ersetzung der vier Gates in den LSTMs durch zwei Gates in den GRUs führt dazu, dass diese einfacher zu implementieren sind und der Berechnungsaufwand geringer ist. Insbesondere werden gegenüber den LSTMs ca. 25% der Parameter eingespart, was gerade bei kleinen Trainingsdatensätzen von Vorteil sein kann.

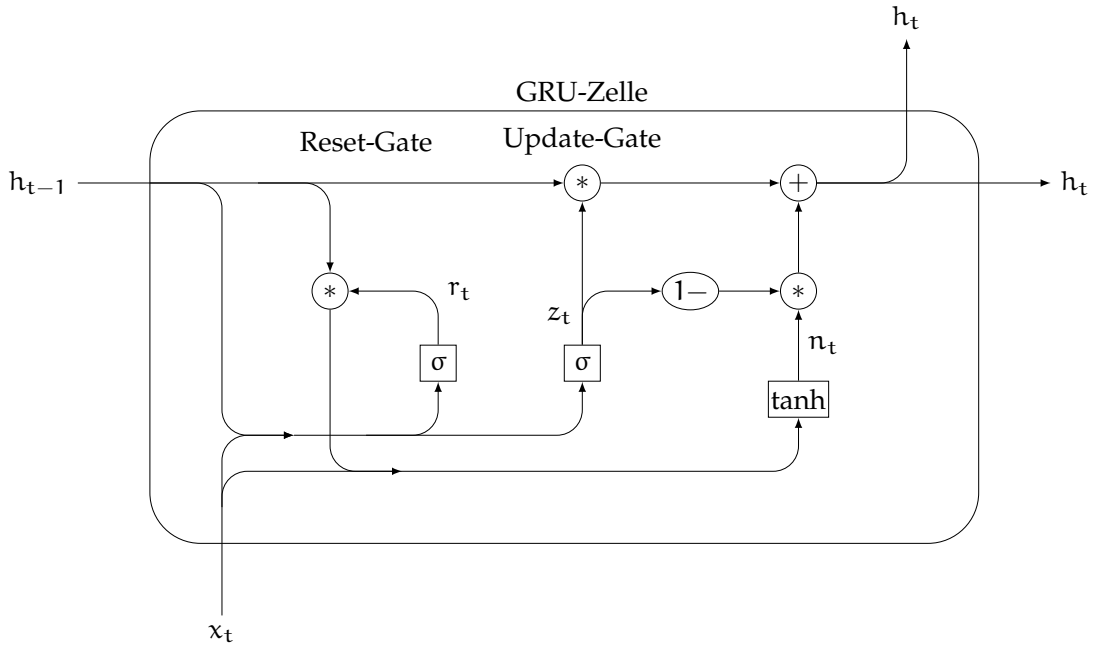


Abbildung 2.5.4: Visualisierung der Rest- und Update-Gates einer GRU Zelle nach [Chr15b]. Die rechteckigen Knoten sind Schichten mit gelernten Parametern, die runden und elliptischen Knoten entsprechen elementweisen Operationen.

2.5.3 *Bidirektionale rekurrente Netze*

Die bisher vorgestellten rekurrenten Neuronalen Netze durchlaufen die Eingabesequenz in ihrer positiven zeitlichen Richtung. Innerhalb der Handschrifterkennung ist dies die Schreibrichtung, also werden Textzeilen des lateinischen Schriftsystems im Normalfall im Sliding-Window Verfahren von links nach rechts durchlaufen. Dadurch kann das Netz zu jedem Zeitpunkt auf die Information vergangener Eingabewerte für die Berechnung der Ausgabe zurückgreifen. Im Fall der Offline-Handschrifterkennung ist das gesamte Wort- oder Zeilenabbild aber vor der Verarbeitung durch das Neuronale Netz gegeben, sodass der zur Verfügung stehende zukünftiger Kontext in dieser Architektur vernachlässigt wird. Der Einbezug dieses Kontextes stellt eine potenzielle Verbesserung der Leistungsfähigkeit des Netzes dar. Soll in der Zukunft liegende Information innerhalb eines normalen RNNs miteinbezogen werden, muss die Ausgabe zeitlich verzögert werden, bis die entsprechenden Zeitschritte beobachtet werden konnten. In der Praxis entsteht allerdings das Problem, dass die korrekte Zeitverzö-

gerung heuristisch festgelegt werden muss und dass sich die Leistungsfähigkeit des Netzes bei großer Verzögerung vermindert. Aus diesen Gründen erweiterten Schuster et al. [SP97] das Konzept der Rekurrenz um die sogenannte Bidirektionalität.

In bidirektionalen Neuronalen Netzen (*Bidirectional Recurrent Neural Networks*, BRNNs) werden die Zellen eines Layers gedoppelt und der positiven bzw. negativen zeitlichen Richtung zugeordnet. Die Ausgaben dieser Zellen werden nach ihrer Berechnung zusammengeführt, sodass die Eingabesequenz parallel in Vorwärts- und Rückwärtsrichtung durchlaufen wird, um die Ausgabe der rekurrenten Schicht zu bestimmen. In Abbildung 2.5.5 ist ein einfaches, entfaltetes bidirektionales rekurrentes Netz dargestellt.

Die Ausgabe eines jeden Zeitschritts ergibt sich aus der zeitlich vorherigen Information durch die Vorwärts-Zellen in Kombination mit der zeitlich folgenden Information durch die Rückwärts-Zellen, sodass zu jedem Zeitpunkt die Information der gesamten Sequenz zur Verfügung steht und der Kontext in beiden Richtungen betrachtet werden kann. Abgesehen von der Zusammenführung der Ausgabe besteht keine Interaktion zwischen den zwei Arten von Zellen. Die für die Backpropagation benötigten Rechenschritte können für die Vorwärts- und Rückwärts-Zellen getrennt durchgeführt werden. Dies ermöglicht das Training der bidirektionalen rekurrenten Netze mit dem bereits genannten Backpropagation Through Time Algorithmus.

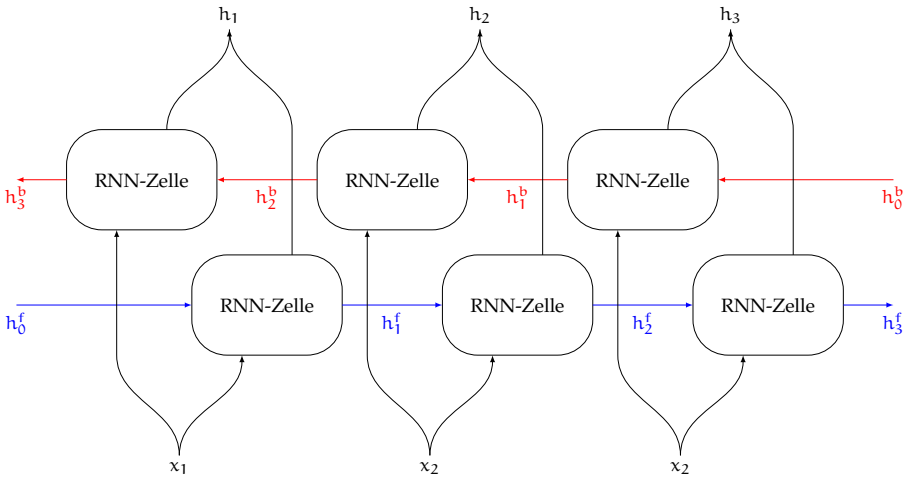


Abbildung 2.5.5: Entfaltetes bidirektionales rekurrentes Netz nach [Chr15a]. Zur Berechnung der Ausgabe h_t werden die Ausgaben $h^f_{(t-1)}$ und $h^b_{(T-t)}$ der Vorwärts- und Rückwärts-Zellen miteinbezogen.

2.5.4 *Multidimensionale rekurrente Netze*

Die bisher vorgestellten rekurrenten Strukturen bewegten sich stets in einer einzelnen Dimension über eine Eingabesequenz. In vielen Anwendungsfällen, wie auch bei der Verarbeitung von Bildern oder Videos, sind die Daten jedoch zwei, drei oder noch höher dimensional. Es können zwar viele Arten von Eingabedaten auf eine eindimensionale Sequenz reduziert werden, beispielsweise indem jeweils eine gesamte Spalte von Pixeln pro Zeitschritt in das Netz eingegeben wird, bei dieser Verarbeitungsart geht jedoch der räumliche Zusammenhang der Daten verloren und das Netz hat keine Möglichkeit mehr, mit Daten beliebiger Höhe umzugehen. In [GFS07] stellen Graves et al. deshalb eine Erweiterung der rekurrenten Struktur auf mehrere Dimensionen vor, die als multidimensionales rekurrentes Neuronales Netz (*Multi-Dimensional Recurrent Neural Network*, MDRNN) bezeichnet wird. Hierbei wird die einzelne rekurrente Verbindung zwischen den Neuronen durch so viele Verbindungen ersetzt, wie die Eingabe Dimensionen besitzt. Dies wird in Abbildung 2.5.6a visualisiert.

Durch die mehrdimensionalen Verbindungen muss die Berechnungsreihenfolge der Neuronen innerhalb einer Schicht eingeschränkt werden, damit die benötigten Ausgaben der in allen Dimensionen zuvor kommenden Neuronen rechtzeitig verfügbar sind, wie in Abbildung 2.5.6b gezeigt wird. Der Punkt (i, j) darf erst nach der Berechnung von $(i-1, j)$ und $(i, j-1)$ erreicht werden. Dafür kann die Sequenz nach Dimensionen geordnet elementweise durchlaufen werden.

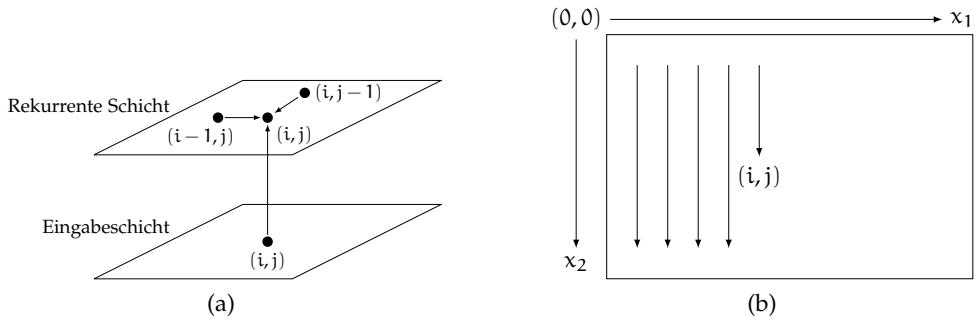


Abbildung 2.5.6: Visualisierungen bezüglich zweidimensionaler rekurrenter Neuronaler Netze nach [GFS07]. (a) Das Neuron (i, j) erhält Information aus der Eingabe an der aktuellen Position (i, j) und dem Kontext der vorherigen Neuronen in beiden Dimensionen $(i-1, j)$ und $(i, j-1)$. (b) Die mehrdimensionalen Neuronenverbindungen erfordern eine angepasste Berechnungsreihenfolge.

Der multidimensionale Ansatz kann, wie auch im eindimensionalen Fall, um Multidirektionalität erweitert werden, was Graves et al. als *Multi-Directional MDRNNs* bezeichnen [GFS07]. Für multidirektionale MDRNNs entstehen dadurch bei n Dimensionen nicht zwei Schichten, die die Sequenz in gegensätzlicher Richtung durchlaufen, sondern 2^n Schichten.

2.6 CONNECTIONIST TEMPORAL CLASSIFICATION

Bei der Verwendung von Neuronalen Netzen für die Schrifterkennung und dem Sequenzlernen im Allgemeinen ergeben sich zwei grundlegende Probleme. Die Länge der zu berechnenden Sequenz ist nicht immer direkt abhängig von der Länge der Eingabesequenz, sondern von den darin enthaltenen Daten. Im Falle der Handschrifterkennung kann das Abbild eines kurzen (wenige Buchstaben), aber breit geschriebenen Wortes breiter sein als das Abbild eines langen (viele Buchstaben) aber schmal geschriebenen Wortes. Aus diesem Grund kann ein klassisches Neuronales Netz nicht direkt die Ausgabesequenz berechnen, da die Länge der Ausgabe dort immer in direktem Zusammenhang mit der Länge der Eingabe, also der Breite des Bildes steht. Zwar kann die Länge der Ausgabesequenz durch die Netzarchitektur, z. B. durch die Verwendung von Faltungsschichten ohne Padding oder Pooling-Schichten, beeinflusst werden, dies geschieht allerdings nicht abhängig vom Bildinhalt, wie es für die Schrifterkennung erforderlich wäre. Außerdem ist die Annotation der weit verbreiteten Datensätze wie der IAM Handwriting Database [MB02] lediglich auf Wort- bzw. Zeilenebene und nicht auf Buchstabenebene verfügbar. Somit kann dem Neuronalen Netz durch den Trainingsprozess nicht vorgegeben werden, an welcher Stelle des Bildes ein Buchstabe geschrieben steht, sondern es ist nur die gesamte gewünschte Ausgabe bekannt. Diese beiden Hindernisse werden durch die Connectionist Temporal Classification gelöst. Die folgenden Unterkapitel orientieren sich an der Darstellung von Graves et al. [GFGS06].

2.6.1 *Netzausgabe und Kostenfunktion*

Bei der Connectionist Temporal Classification (CTC) [GFGS06] werden die Ausgaben des Neuronalen Netzes als Wahrscheinlichkeitsverteilung über ein Alphabet für alle Zeitschritte betrachtet. Konkret bedeutet dies, dass das Netz bei einer Eingabesequenz $\mathbf{x}$ der Länge T mit Zeichen aus einem Alphabet L eine Ausgabesequenz $\mathbf{y}$ der Länge T erzeugt, die für jeden Zeitschritt einer Wahrscheinlichkeitsverteilung über das Alphabet $L' = L \cup \{-\}$ entspricht. Mit $-$ wird das sogenannte CTC-Blank Symbol

bezeichnet. Es muss zwischen Pfad und Transkription unterschieden werden. Ein Pfad $\pi \in L'^T$ ist eine Sequenz von Symbolen aus L', deren bedingte Wahrscheinlichkeit bei Betrachtung eines Musters $\mathbf{x}$

$$p(\pi|\mathbf{x}) = \prod_{t=1}^{T} y^t_{\pi_t}, \forall \pi \in L'^T \tag{2.6.1}$$

mithilfe der Ausgaben $y^t_{\pi_t}$ des Netzes berechnet werden kann. Eine Transkription $\mathbf{l} \in L^{\leqslant T}$ ist eine Sequenz von Symbolen aus dem Alphabet L mit einer Länge $|\mathbf{l}| \leqslant T$. Die Transkription $\mathbf{l}$ wird aus π durch eine Abbildung $\mathcal{B} : L'^T \mapsto L^{\leqslant T}$ gewonnen, die zunächst alle doppelten Symbole und dann alle CTC-Blank Symbole aus dem Pfad entfernt. Dies erklärt auch die Notwendigkeit des CTC-Blank Symbols. Zum einen können doppelte Buchstaben in der Transkription durch eine Trennung mithilfe eines Blank Symbols erzeugt werden und zum anderen kann eine lange Eingabesequenz (beispielsweise eines kurzen aber breit geschriebenen Wortes) durch die Verwendung von doppelten Elementen und Blanks in eine kürzere Ausgabesequenz überführt werden. Eine Transkription kann im Allgemeinen mehrere mögliche Pfade besitzen (z. B. $\mathcal{B}(aa-abb) = \mathcal{B}(a-aabbb) = aab$), die diese durch die Abbildung $\mathcal{B}$ erzeugen, sodass die resultierende Wahrscheinlichkeit einer Transkription durch die Summe ihrer Pfade

$$p(\mathbf{l}|\mathbf{x}) = \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{l})} p(\pi|\mathbf{x}) \tag{2.6.2}$$

gegeben ist. Die Pfade, die eine gegebene Transkription erzeugen, werden dabei durch die Inverse der Abbildung $\mathcal{B}$ gewonnen. Das Training soll die Wahrscheinlichkeit der korrekten Transkription und damit die Wahrscheinlichkeit dazugehöriger Pfade maximieren. Es soll also ein Klassifikator

$$h(\mathbf{x}) = \arg\max_{\mathbf{l} \in L^{\leqslant T}} p(\mathbf{l}|\mathbf{x}). \tag{2.6.3}$$

konstruiert werden. Hierfür wird eine differenzierbare Zielfunktion benötigt, die die Wahrscheinlichkeit korrekter Pfade π maximiert und die damit das Training des Neuronalen Netzes mithilfe von Backpropagation (Through Time) ermöglicht. Die bei der Connectionist Temporal Classification verwendete Zielfunktion ist

$$C_{ML}(S, \mathcal{N}_{\boldsymbol{w}}) = -\sum_{(\mathbf{x},\mathbf{l}) \in S} \ln(p(\mathbf{l}|\mathbf{x})), \tag{2.6.4}$$

wobei der Datensatz S einer Menge von Beispielen mit Muster $\mathbf{x}$ und Transkription $\mathbf{l}$ entspricht und $\mathcal{N}_{\boldsymbol{w}}$ ist das Netz mit Parametern $\boldsymbol{w}$. Sie ermöglicht das Training

des Neuronalen Netzs ohne eine Annotation auf Buchstabenebene, da gleichzeitig die Wahrscheinlichkeiten aller korrekten Netzausgaben unabhängig von den Buchstabenpositionen oder -wiederholungen maximiert werden. Ein Problem besteht in der effizienten Berechnung der bedingten Wahrscheinlichkeiten $p(\mathbf{l}|\mathbf{x})$ (siehe Gleichung 2.6.2). Die Summe wird über alle Pfade für eine bestimmte Transkription gebildet, von denen im Allgemeinen viele existieren. Dieses Problem kann aber durch dynamische Programmierung, ähnlich dem Forward-Backward Algorithmus für Hidden Markov Modelle, gelöst werden.

2.6.2 *CTC Forward-Backward Algorithmus*

Für die Berechnung der Kostenfunktion 2.6.4 wird eine effiziente Art der Berechnung der Produktionswahrscheinlichkeit $p(\mathbf{l}|\mathbf{x})$ der Transkription $\mathbf{l}$ gegeben des Musters $\mathbf{x}$ benötigt, die mithilfe der Vorwärts- und Rückwärtsvariablen α und β gegeben ist

$$p(\mathbf{l}|\mathbf{x}) = \sum_{s=1}^{|\mathbf{l}'|} \frac{\alpha_t(s)\beta_t(s)}{y_{\mathbf{l}'_s}^t}. \tag{2.6.5}$$

Die Vorwärtsvariable $\alpha_t(s)$ entspricht der bedingten Wahrscheinlichkeit von $\mathbf{l}_{1:s}$, also dem Auftreten der ersten s Symbole in $\mathbf{l}$ zum Zeitpunkt t

$$\alpha_t(s) = \sum_{\substack{\pi \in L'^T: \\ \mathcal{B}(\pi_{1:t}) = \mathbf{l}_{1:s}}} \prod_{t'=1}^{t} y_{\pi_{t'}}^{t'}. \tag{2.6.6}$$

Dafür wird über alle Pfadwahrscheinlichkeiten summiert, die mit Pfad-Präfixen $\pi_{1:t}$ korrespondieren, deren zugehörige Transkription die ersten s Zeichen von $\mathbf{l}$ ergibt. Um CTC-Blanks in den Pfaden zulassen zu können, wird eine erweiterte Sequenz $\mathbf{l}'$ mit $|\mathbf{l}'| = 2|\mathbf{l}| + 1$ betrachtet, bei der vor den Anfang und hinter das Ende sowie zwischen allen Paaren verschiedener Symbole Blanks hinzugefügt werden. Die Vorwärtsvariablen α können dann mit der rekursiven Vorschrift

$$\begin{aligned}
\alpha_1(1) &= y_b^1 \\
\alpha_1(2) &= y_{\mathbf{l}_1}^1 \\
\alpha_1(s) &= 0, \forall s > 2
\end{aligned}$$

$$\alpha_t(s) = \begin{cases} (\alpha_{t-1}(s) + \alpha_{t-1}(s-1)) y_{\mathbf{l}'_s}^t \text{ wenn } \mathbf{l}'_s = b \vee \mathbf{l}'_{s-2} = \mathbf{l}'_s \\ (\alpha_{t-1}(s) + \alpha_{t-1}(s-1)) + \alpha_{t-1}(s-2) y_{\mathbf{l}'_s}^t \text{ sonst} \end{cases} \tag{2.6.7}$$

berechnet werden. Die Wahrscheinlichkeit von $\mathbf{l}$ entspricht der Summe der Wahrscheinlichkeiten von $\mathbf{l}'$ mit und ohne dem letzten Blank Symbol $p(\mathbf{l}|\mathbf{x}) = \alpha_T(|\mathbf{l}'|) + \alpha_T(|\mathbf{l}'| - 1)$. Die Rückwärtsvariablen β haben eine duale Funktion. Sie beschreiben die Wahrscheinlichkeit des Auftretens von $\mathbf{l}_{s:|\mathbf{l}|}$ der letzten $|\mathbf{l}| - s$ Symbole zum Zeitpunkt t

$$\beta_t(s) = \sum_{\substack{\pi \in L'^T: \\ \mathcal{B}(\pi_{t:T}) = \mathbf{l}_{s:|\mathbf{l}|}}} \prod_{t'=t}^{T} y^{t'}_{\pi_{t'}} \tag{2.6.8}$$

$$\begin{aligned} \beta_T(|\mathbf{l}'|) &= y^T_b \\ \beta_T(|\mathbf{l}'| - 1) &= y^T_{\mathbf{l}_{|\mathbf{l}|}} \\ \beta_T(s) &= 0, \forall s < |\mathbf{l}'| - 1 \end{aligned}$$

$$\beta_t(s) = \begin{cases} (\beta_{t+1}(s) + \beta_{t+1}(s+1)) y^t_{\mathbf{l}'_s} \text{ wenn } \mathbf{l}'_s = b \vee \mathbf{l}'_{s+2} = \mathbf{l}'_s \\ (\beta_{t+1}(s) + \beta_{t+1}(s+1) + \beta_{t+1}(s+2)) y^t_{\mathbf{l}'_s} \text{ sonst.} \end{cases} \tag{2.6.9}$$

Aus Gleichungen 2.6.6 und 2.6.8 ergibt sich

$$\alpha_t(s)\beta_t(s) = \sum_{\substack{\pi \in \mathcal{B}^{-1}(\mathbf{l}): \\ \pi_t = \mathbf{l}'_s}} y^t_{\mathbf{l}'_s} \prod_{t=1}^{T} y^t_{\pi_t} \tag{2.6.10}$$

und mit Gleichung 2.6.1

$$\frac{\alpha_t(s)\beta_t(s)}{y^t_{\mathbf{l}'_s}} = \sum_{\substack{\pi \in \mathcal{B}^{-1}(\mathbf{l}): \\ \pi_t = \mathbf{l}'_s}} p(\pi|\mathbf{x}). \tag{2.6.11}$$

Dies entspricht dem Teil der Wahrscheinlichkeit $p(\mathbf{l}|x)$ der Pfade, die zum Zeitpunkt t durch $\mathbf{l}'_s$ gehen. Für ein beliebiges t kann demnach über alle Elemente s der Transkription $\mathbf{l}'$ summiert werden, um die Produktionswahrscheinlichkeit $p(\mathbf{l}|\mathbf{x})$ (siehe Gleichung 2.6.2) zu erhalten. Somit kann die Kostenfunktion 2.6.4 bei gegebener Transkription $\mathbf{l}$ effizient berechnet werden. Für den praktischen Einsatz eines CTC-Netzwerks fehlt allerdings eine ebenso effiziente Methode zur Berechnung der wahrscheinlichsten Transkription für ein gegebenes Muster.

2.6.3 *Dekodierung*

Die Berechnung der wahrscheinlichsten Transkription wird in Anlehnung an die Terminologie von Hidden Markov Modellen als Dekodierung bezeichnet [GFGS06, Fin14]. Es ist kein effizientes Verfahren zur exakten Berechnung von Gleichung 2.6.3 bekannt, aber es existieren mehrere Approximationen. Eine triviale Methode heißt *Best Path Decoding* und berechnet eine Transkription durch die Abbildung des wahrscheinlichsten Pfades π^* auf eine Transkription durch $h(\mathbf{x}) \approx \mathcal{B}(\pi^*)$. Für die Bestimmung des wahrscheinlichsten Pfades wird zu jedem Zeitpunkt das wahrscheinlichste Element des Alphabets gewählt, wie in Abbildung 2.6.1 zu sehen ist. Da eine andere Transkription existieren kann, die zwar mit Pfaden geringerer Wahrscheinlichkeit korrespondiert, deren summierte Wahrscheinlichkeit aber größer ist als $p(\pi^*|\mathbf{x})$, ist die Optimalität der Lösung vom Best Path Decoding nicht garantiert.

Beim *Prefix Search Decoding* wird die Tatsache genutzt, dass über eine Modifikation des Forward-Backward Algorithmus die Wahrscheinlichkeit sukzessiver Erweiterungen eines Transkriptions-Präfixes berechnet werden kann. Da die Anzahl der Präfixe jedoch exponentiell mit der Sequenzlänge wächst, werden weitere Heuristiken für die effiziente Berechnung benötigt. Die Idee von Graves et al. basiert darauf, dass die Ausgaben für Buchstaben eines trainierten CTC-Netzwerks im Normalfall von starken Blank-Prädiktionen umgeben sind. Das Prefix Search Decoding kann mithilfe eines Schwellwerts auf jede Teilsequenz angewendet werden, die mit einem Blank beginnt und endet um die Anzahl der zu betrachtenden Präfixe zu reduzieren [GFGS06].

Eine weitere Variante des Prefix Search Decodings ist *Beam Search*. Das exponentielle Wachstum der zu expandierenden Präfixe wird durch die Verwendung einer Strahlbreite reduziert. Präfixe, die einen gewissen Anteil der Wahrscheinlichkeit des besten Präfixes unterschreiten, werden verworfen und nicht weiter verfolgt [Fin14, S. 186ff].

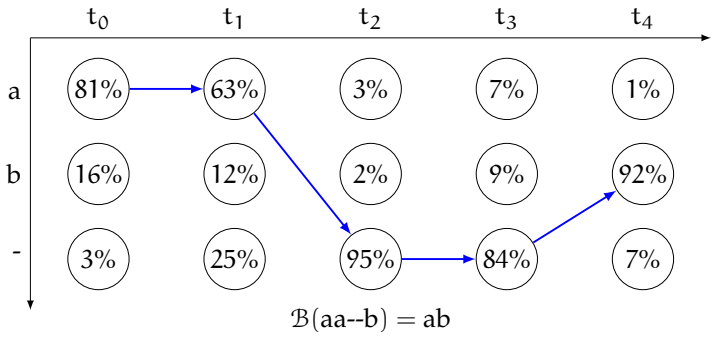


Abbildung 2.6.1: Visualisierung des Best Path Decodings der Connectionist Temporal Classification für eine Sequenz mit einem Alphabet $L' = \{a, b, -\}$ nach [Sch18].

3 VERWANDTE ARBEITEN

Die Handschrifterkennung hat eine lange methodische Geschichte, mit vielen verschiedenen Entwicklungen und Ansätzen. Hierzu zählen unter anderem Verfahren mit Hidden Markov Modellen sowie Neuronalen Netzen. Das erste Faltungsnetz LeNet [LBBH98] wurde beispielsweise für das Erkennen von handschriftlich geschriebenen Zahlen entwickelt. Ein zentrales Problem der Handschrifterkennung ist Sayres Paradoxon. Es besagt, dass die Erkennung von handschriftlich geschriebenen Wörtern einer Segmentierung bedarf und die Segmentierung von Wörtern ohne ihre Erkennung schwer möglich ist [Say73]. Aus diesem Grund beschränkten sich frühe Verfahren auf die Erkennung einzelner Wörter. Heute werden meist Methoden wie Hidden Markov Modelle oder rekurrente Neuronale Netze verwendet, die eine integrierte Segmentierung und Klassifikation durchführen können. Im Folgenden soll eine Auswahl etablierter Verfahren präsentiert werden, die einen Einfluss auf die Entwicklung der in Kapitel 4 vorgestellten Methodik hatten, oder die Alternativen dazu darstellen.

3.1 TIEFE FALTUNGSNETZE

Eine für die gesamte Forschungsrichtung der Mustererkennung und insbesondere der Bildverarbeitung essenzielle Entwicklung ist die der tiefen Faltungsnetze. Noch heute orientieren sich Faltungsnetze typischerweise an der Architektur des *LeNet* von LeCun et al. [LBBH98]. Sie führt eine Merkmalsextraktion mit einer hierarchischen Anordnung von Faltungs- und Subsampling (Unterabtastungs-) Schichten durch, denen ein Klassifikator aus einer Menge von voll-verbundenen Schichten folgt. Diese Struktur wird in Abbildung 3.1.1 dargestellt. Faltungsschichten zeigen sich in den Versuchen von LeCun et al. als robust gegenüber Translationen und Skaleninvarianzen und zeichnen sich durch ihre geringe Parameteranzahl im Vergleich zu voll-verbundenen Schichten aus. Besonders bei hoch-dimensionalen Eingabedaten (z. B. Bildern) ermöglichen Faltungsschichten somit erst die Verarbeitung durch Neuronale Netze mit mehreren Schichten, insbesondere mit der zum Zeitpunkt der Veröffentlichung verfügbaren Hardware. Mit diesem Vorgehen konnte eine Fehlerrate auf dem MNIST Datensatz [LBBH98] für die Klassifikation handschriftlicher Ziffern von 0.95% erzielt werden.

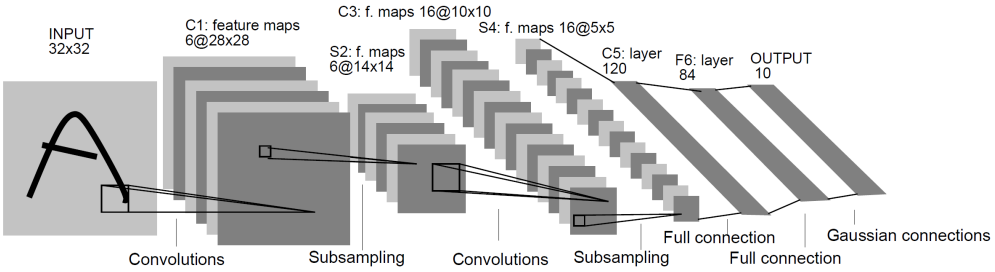


Abbildung 3.1.1: Darstellung der LeNet-Architektur für die Erkennung von Ziffern aus [LBBH98]. Eine Mischung aus Faltungs- und Pooling-Schichten extrahiert Merkmale aus dem Eingabebild, die mit drei voll-verbundenen Schichten klassifiziert werden.

Die erste erfolgreiche Netzarchitektur in der ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [RDS+15] für die Bildklassifikation ist das *AlexNet* von Krizhevsky et al. [KSH12]. Die Verfügbarkeit des ImageNet Datensatzes mit 1.2 Millionen Trainingsbeispielen eröffnet die Möglichkeit des Trainings tiefer Neuronaler Netze mit einer großen Anzahl von Parametern. Im Falle des AlexNet überstieg die Größe ihrer Netzarchitektur den Grafikkartenspeicher von 3 GB, sodass das Netz auf zwei Grafikkarten aufgeteilt wurde, wie in Abbildung 3.1.2 zu erkennen ist. Es besteht aus fünf Faltungsschichten mit Kernelgrößen von 11×11 und 5×5 in den ersten beiden Schichten und 3×3 für die letzten drei Schichten. Auch die darauf folgenden drei voll-verbundenen Schichten werden jeweils zur Hälfte auf den beiden Grafikkarten berechnet und gespeichert. Trotz des großen Umfangs an Trainingsmaterial wurden die Eingabedaten augmentiert und Dropout in den ersten beiden voll-verbundenen Schichten angewendet, um Overfitting zu reduzieren. Die Autoren erzielten mit einem Ensemble aus fünf solcher Faltungsnetze im ILSVRC-2012 Wettbewerb eine Top-5 Klassifikationsfehlerrate von 16.4% und übertrafen damit das zweitplatzierte Verfahren von Harada et al. [HK12] um 9.9%. In der Folge dieses Erfolgs vervierfachte sich die Anzahl der Teilnehmer für den Wettbewerb im Folgejahr von 6 auf 24, von denen eine Mehrheit auf tiefe Neuronale Netze setzte [RDS+15].

Eine weitere einflussreiche Faltungsnetzarchitektur ist das *VGGNet* von Simonyan und Zisserman [SZ14]. Der hauptsächliche Unterschied zwischen dem VGGNet und AlexNet besteht, neben der größeren Anzahl von 16 bis 19 Schichten, in der konsequenten Verwendung von Rezeptiven Feldern der Größe 3×3. Mehrere hintereinander angeordnete Faltungsschichten vergrößern das effektive Rezeptive Feld der hintersten Schicht bezogen auf die Eingabe, ohne dass die Kernelgröße innerhalb der Schichten

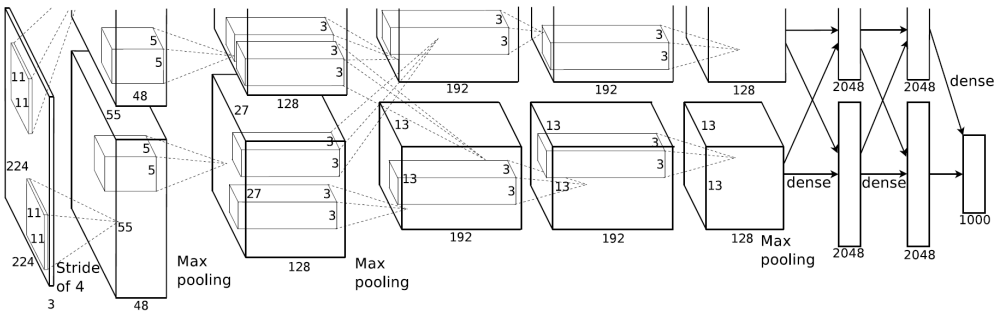


Abbildung 3.1.2: Architektur des AlexNet aus [KSH12]. Das Netz, bestehend aus fünf Faltungs- und drei voll-verbundenen Schichten, wird aufgrund des limitierten Speichers auf zwei Grafikkarten aufgeteilt, zwischen denen zur Erhöhung der Berechnungsgeschwindigkeit nur partiell eine Kommunikation stattfindet.

vergrößert werden muss. Damit wird die Anzahl der Parameter reduziert, da drei Faltungsschichten mit einem Rezeptiven Feld der Größe 3×3 und C Ein- und Ausgabekanälen zusammen $27C^2$ Gewichte besitzen, während eine einzelne Faltungsschicht der Größe 7×7 insgesamt $49C^2$ Parameter umfassen würde. Außerdem wird die nicht-lineare Aktivierungsfunktion auf diese Weise mehrmals angewendet, was laut den Autoren in einer diskriminativeren Entscheidungsfunktion resultieren soll. Die VGGNet Variante E mit insgesamt 19 Schichten, die in Abbildung 3.1.3 dargestellt wird, besteht aus 16 Faltungsschichten mit steigender Anzahl von Kanälen für die Merkmalsextraktion und drei voll-verbundene Schichten für die Klassifikation. Die Autoren erreichten mit dieser Architektur den zweiten Platz im ILSVRC-2014 Wettbewerb mit einer Top-5 Fehlerrate von 7.3% hinter der GoogLeNet genannten Architektur von Szegedy et al. [SLJ+15] mit 6.7%.

3.2 NN-HMM KOMBINATIONEN

Durch ihren verbreiteten und erfolgreichen Einsatz innerhalb der Spracherkennung, fanden Hidden Markov Modelle (HMMs) ab den 1980er Jahren auch innerhalb der Handschrifterkennung Anwendung [NWF86, KHB88]. Die generative Struktur, bestehend aus Zuständen mit Übergängen und Emissionen, ermöglicht es HMMs eine integrierte Segmentierung und Klassifikation der handgeschriebenen Wörter und Zeilen vorzunehmen, ohne dass segmentierte Trainingsdaten vorliegen müssen. Neuronale Netze zeigen eine hohe Leistungsfähigkeit bei der Verarbeitung von Bilddaten [LBBH98, KSH12], was ihren Einsatz auch innerhalb der Sprach- und Handschrifter-

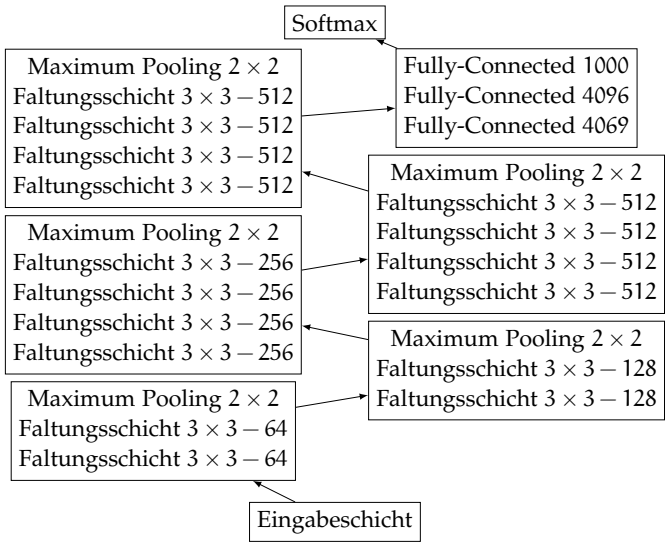


Abbildung 3.1.3: Faltungsnetzarchitektur VGG16E von [SZ14]. Bei Faltungsschichten wird die Kernelgröße (3×3) und die Anzahl von Filtern je Schicht angegeben. Beim Maximum Pooling ist die Kernelgröße der einzige Parameter. Bei vollverbundenen Schichten wird die Anzahl von Neuronen je Schicht angegeben. Bei Softmax-Schichten wenden die Neuronen elementweise Operationen an.

kennung motiviert. Klassische Neuronale Netze vor der Entwicklung der Connectionist Temporal Classification unterliegen jedoch der in Kapitel 2.6 beschriebenen Problematik von verschiedenen Längen der Ein- und Ausgabesequenzen sowie der fehlenden Segmentierung auf der Buchstabenebene der Trainingsdaten. Die Kombination von Neuronalen Netzen mit Hidden Markov Modellen innerhalb des *Hybrid Approach* [BM12] stellt deshalb einen interessanten Ansatz dar, der die hohe Leistungsfähigkeit der Neuronalen Netze mit den Möglichkeiten der HMMs verbinden soll.

Innerhalb des hybriden Ansatzes werden Neuronale Netze dazu verwendet die a-posteriori Zustandswahrscheinlichkeiten der HMMs zu berechnen [VDW+15]. Somit ersetzen die Neuronalen Netze in diesem Ansatz die zuvor heuristisch gewählten Merkmale der HMMs. Es existieren verschiedene Variationen von hybriden Ansätzen der NN-HMMs, die sich unter anderem in der verwendeten Art vom Neuronalen Netz, HMM Parametern und den Trainingsalgorithmen unterscheiden.

Bengio et al. setzen für ihre *OUTSEG* Architektur ein Faltungsnetz in Verbindung mit einem HMM mit drei Zuständen pro Buchstabe für die Online-Handschrifterkennung ein [BLNB95]. Das Ablaufschema des Verfahrens ist in Abbildung 3.2.1 zu sehen. Eine Besonderheit stellt dabei die *AMAP* Eingaberepräsentation dar. Hier wird die

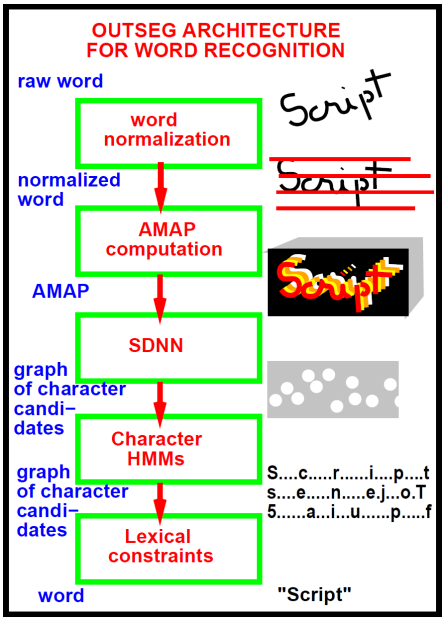


Abbildung 3.2.1: Schema der OUTSEG-Architektur aus [BLNB95].

Stift-Trajektorie mithilfe ihrer lokalen Eigenschaften in einem niedrig aufgelöstem Bild dargestellt. Die Autoren erzielten eine Zeichenfehlerrate (siehe Kapitel 5.2) von 2.0% auf einem eigens erstellen Datensatz mit 881 kleingeschriebenen Wörtern, wobei die Ausgabe durch ein Lexikon mit ca. 25 500 Wörtern beschränkt war.

Schenkel et al. [SGH95] verwenden sogenannte *Time-Delay Neural Networks* (TDNN) für die Online-Handschrifterkennung. Diese berechnen mit einer heuristisch gewählten Merkmalsrepräsentation und einem temporal eingeschränktem rezeptiven Feld der Neuronen Wahrscheinlichkeiten für das Vorkommen der Buchstaben im jeweiligen Zeitfenster. Damit hat das HMM vor allem die Aufgabe die Dauer (Breite) der Buchstaben des Wortes anhand der Ausgabe des Netzes zu modellieren. Das Training des NN findet dabei unabhängig vom HMM statt, indem eine Schicht eingefügt wird, die lediglich das Vorkommen von Buchstaben im Wort bestimmt. Somit ist das Training unabhängig von der Buchstabenposition und -reihenfolge und das Netz kann mithilfe der wortbasierten Annotation optimiert werden. Der Trainings- und Testdatensatz wurde dabei wiederum von den Autoren erstellt, in diesem Fall auf der Basis von händischen Abschriften von Textauszügen aus dem Wall Street Journal. Hier konnte eine Zeichenfehlerrate von 4.2% erreicht werden.

Auch in [KMB+10] wird ein Neuronales Netz, in diesem Fall ein MLP, verwendet, um a-posteriori Wahrscheinlichkeiten zu bestimmen. Hier geschieht dies jedoch auf Sub-Buchstaben Ebene (*Grapheme-based*). Es existiert eine zyklische Abhängigkeit zwischen der Annotation auf Merkmalsebene und den a-posteriori Wahrscheinlichkeiten. Die Annotation wird durch das HMM berechnet und vom Netz für das Training benötigt; umgekehrt basiert das HMM jedoch auf den vom Netz bestimmten a-posteriori Wahrscheinlichkeiten. Daher wird das Training des NNs und HMMs, vergleichbar mit dem *Expectation Maximization* Algorithmus, abwechselnd durchgeführt. Im ICDAR2009 Wettbewerb [GEA09] mit dem RIMES Datensatz [ACG+06] hatte das Verfahren auf diese Weise ohne die Berücksichtigung von Groß- und Kleinschreibung eine Wortfehlerrate von 18.6% mit einem Lexikon, in dem nur die 1 612 Wörter des Test-Datensatzes enthalten sind und eine Wortfehlerrate von 24.3% mit einem Lexikon von 5 334 Wörtern aus dem Trainings- und Test-Datensatz.

Voigtlaender et al. [VDW+15] setzen ein bidirektionales LSTM in Kombination mit einem HMM mit einer zustandsabhängigen Topologie ein. Eine Besonderheit ist hier das sogenannte *Sequence-discriminative Training*, bei dem das *Maximum Mutual Information* Kriterium anstelle der Kreuzentropie verwendet wird. Hiermit werden auf dem IAM Datensatz Wort- und Zeichenfehlerraten von 12.7% bzw. 4.8% und auf dem RIMES Datensatz von 12.1% bzw. 4.3% erzielt.

Die Verwendung des Forward-Backward Algorithmus für das CTC Training legt die Verwandschaft der CTC und des hybriden Ansatzes nahe. Théodore Bluche untersucht in seiner Dissertation unter anderem die Unterschiede und Gemeinsamkeiten bei der Verwendung von Neuronalen Netzen für die Handschrifterkennung sowohl im hybriden Ansatz mit HMMs als auch der Connectionist Temporal Classification [Blu15, S. 171]. Er stellt den Vergleich an, dass die Connectionist Temporal Classification dem hybriden Ansatz mit einem HMM mit einer eingeschränkten Architektur (ein Zustand pro Buchstaben und Blank Symbol) und ohne Zustandsübergangswahrscheinlichkeiten und a-priori Zustandswahrscheinlichkeiten entspricht.

3.3 ATTRIBUTRAUM-REPRÄSENTATIONEN

Eine weitere Variante der Handschrifterkennung kann durch die Abbildung eines Wortabbildes und möglicher Transkriptionen in einen gemeinsamen Attributraum durchgeführt werden. Diese sogenannten *Attribute Space* Verfahren verwenden ein Lexikon bekannter Wörter, die in den Attributraum transformiert werden. Nach der Berechnung der Attribut-Repräsentation für ein Wortabbild kann die Erkennung mithilfe des Nächsten-Nachbar Verfahrens durchgeführt werden, indem das Wortabbild

im Attributraum mithilfe eines Distanzmaßes dem am nächsten liegenden Wort im Lexikon zugeordnet wird. Eine mögliche Repräsentation sind die *Pyramidal Histogram of Characters* (PHOC) [AGFV14]. Bei der PHOC-Repräsentation werden die Vorkommen von Buchstaben in bestimmten Teilen des Wortes oder Wortabbildes binär kodiert. Ein einfaches binäres Buchstabenhistogramm reicht zur Repräsentation eines Wortes nicht aus, da beispielsweise die Wörter *silent* und *listen* ein identisches Histogramm besitzen. Bei den PHOCs werden deshalb Histogramme in mehreren Ebenen pyramidal aufgebaut, wobei das Wort auf Ebene i in i Teile aufgeteilt wird, für die jeweils ein separates Histogramm berechnet wird. Die PHOC Repräsentation ergibt sich dann aus der Konkatenation aller Histogramme auf allen Ebenen, wie in Abbildung 3.3.1 zu sehen ist. Almazan et al. [AGFV14] fügen noch eine weitere Ebene mit den 50 häufigsten Bigrammen hinzu, sodass sich mit den Ebenen zwei bis fünf und Bigrammen bei einem Alphabet mit 36 Buchstaben und Zahlen ein 604-dimensionaler Attributraum ergibt. Zur Verwendung der PHOCs im Word-Spotting oder der Handschrifterkennung wird eine Methode benötigt, um aus einem gegebenen Wortabbild die PHOC Repräsentation zu bestimmen. Dafür werden oft tiefe Neuronale Netze eingesetzt.

Poznanski und Wolf [PW16] verwenden Faltungsnetze mit einer Architektur, die vom VGGNet [SZ14] inspiriert ist. Nach dem gemeinsamen Faltungsteil wird für jede Attributgruppe (erster, zweiter, ... Teil des Wortes) auf jeder Ebene ein eigener Satz von drei voll-verbundenen Schichten eingesetzt. Die Faltungsschichten können so die allgemeine Gestalt und die Merkmale von Buchstaben erfassen und die voll-verbundenen Schichten spezialisieren sich auf die Lokalisation eines bestimmten Zeichens. Da die voll-verbundenen Schichten eine feste Eingabegröße besitzen, skalieren und verzerren die Autoren das Wortabbild auf eine Größe von 100×32 Pixel. Die Netzarchitektur wird in Abbildung 3.3.2 dargestellt. Vor dem Nächsten-Nachbar Vergleich zwischen der Ausgabe des Netzes und den PHOC-Repräsentationen der Wörter des Lexikons werden alle Elemente mit einer kanonischen Korrelationsanalyse in einen gemeinsa-

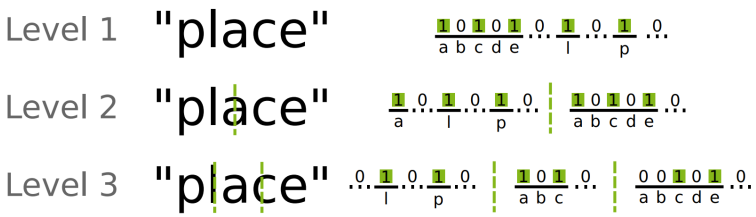


Abbildung 3.3.1: Visualisierung der Pyramidal Histogram of Characters aus [SF16].

men Unterraum projiziert. Mit dieser Methodik konnten die Autoren eine Zeichen- und Wortfehlerrate von 3.44% bzw. 6.45% auf dem IAM Datensatz und von 1.9% bzw. 3.9% auf dem RIMES Datensatz erreichen.

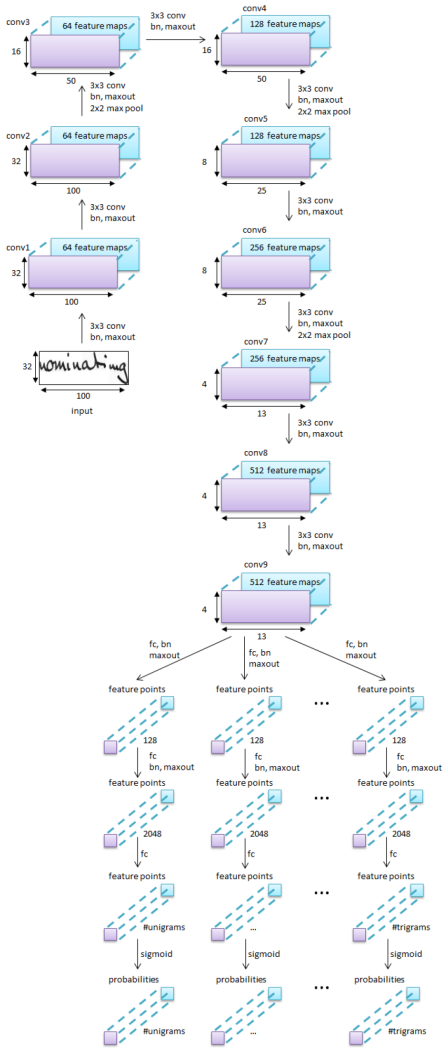


Abbildung 3.3.2: Visualisierung der CNN-N-Gramm Architektur aus [PW16]. Insgesamt neun Faltungsschichten extrahieren Merkmale aus dem Eingabebild. In 19 parallelen Zweigen werden aus den Merkmalen der Faltungsschichten die einzelnen PHOC-Attributgruppen berechnet.

Ein ähnliches Vorgehen wird von Sudholt und Fink in [SF16] für Word-Spotting beschrieben. Auch hier wird eine CNN Architektur, genannt PHOCNet, dazu eingesetzt eine PHOC Repräsentation von Wortabbildern zu berechnen. Im Gegensatz zu der Architektur von Poznanski und Wolf wird die Eingabegrößenbeschränkung der voll-verbundenen Schichten jedoch durch die Verwendung einer *Spatial Pyramid Pooling* (SPP) Schicht [HZRS15] umgangen. Diese SPP-Schicht wird zwischen den Faltungs- und voll-verbundenen Schichten eingefügt und berechnet einen Merkmalsvektor fester Dimensionalität aus den Eingabedaten verschiedener Größe. Indem die Pooling-Operation in lokalen Bereichen der Merkmalskarte durchgeführt wird, die eine zur Eingabe proportionale Größe haben, bleibt dabei ein großer Teil der örtlichen Information der Merkmale erhalten. Im Gegensatz zu Poznanski et al. werden lediglich drei hintereinander geschichtete voll-verbundene Schichten für die Berechnung aller Attributgruppen der PHOC Repräsentation verwendet, sodass keine parallelen Pfade in der Netzarchitektur benötigt werden, wie in Abbildung 3.3.3 zu sehen ist. Wie bei Poznanski und Wolf wird für die Ausgabeschicht eine Sigmoid Aktivierungsfunktion und die Kreuzentropie Kostenfunktion eingesetzt, da die Berechnung der binären PHOC-Vektoren als eine Multi-Label Klassifikation angesehen werden kann. Dieses Modell wurde jedoch lediglich für den Word-Spotting Anwendungsfall und nicht für die Transkription von Handschrift evaluiert.

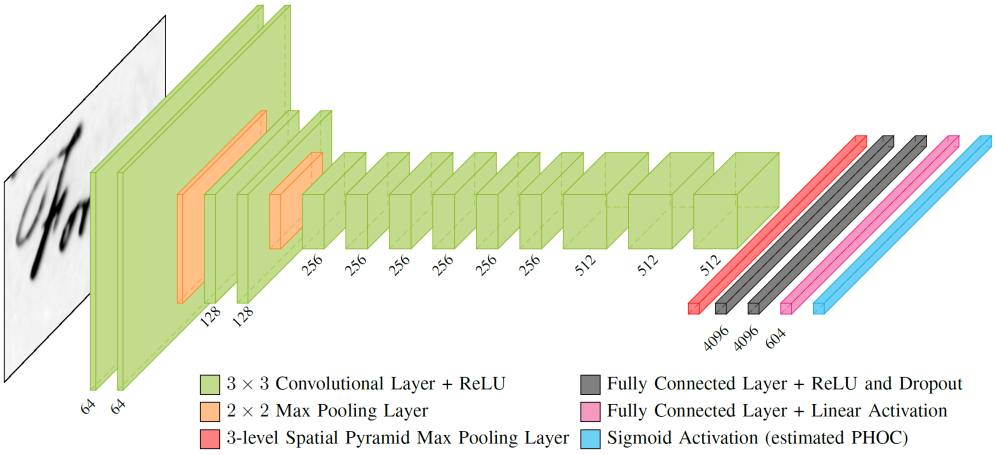


Abbildung 3.3.3: Darstellung der PHOCNet Architektur aus [SF16]. Eine Spatial Pyramid Pooling-Schicht berechnet aus der Ausgabe der Faltungsschichten eine Merkmalsrepräsentation konstanter Größe, die durch drei voll-verbundene Schichten in die PHOC Ausgaberepräsentation überführt wird.

3.4 LSTM-CTC KOMBINATIONEN

Mit der in Kapitel 2.6 beschriebenen Connectionist Temporal Classification können Neuronale Netze auch ohne die Verwendung von hybriden Architekturen mit Hidden Markov Modellen für das Sequenzlernen eingesetzt werden. Zusammen mit der Einführung des CTC-Verfahrens [GFGS06] stellen Graves et al. Ergebnisse für die Spracherkennung auf dem TIMIT Sprachdatensatz [GLF+93] vor. Sie setzen ein einfaches Netzwerk bestehend aus einer bidirektionalen LSTM-Schicht und einer Ausgabeschicht ein. In Kombination mit der Connectionist Temporal Classification konnten die Ergebnisse unter der Verwendung von 26 Merkmalen für jedes 10 Millisekunden breite Fenster gegenüber dem klassischen hybriden Ansatz um ca. 1% auf eine Label Fehlerrate von 30.51% verbessert werden.

Für die Handschrifterkennung ist das CTC-Verfahren erstmals im Jahr 2008 von Graves et al. verwendet worden [GLF+08]. Die gleiche Architektur, die auch für die Spracherkennung eingesetzt wurde, übertraf sowohl in der Online- (IAM-OnDB [LB05]) als auch Offline-Handschrifterkennung (IAM-DB [MB02]) die Ergebnisse des hybriden Ansatzes mit vergleichbarer RNN Architektur um mehr als 10%. In beiden Fällen wurden, ähnlich zur Spracherkennung, heuristische Merkmale verwendet, die auf den Daten mit dem Sliding-Window Verfahren berechnet wurden und dem Netz als Eingabe dienen. Für einen Teil der Experimente wurde außerdem ein Sprachmodell auf der Basis von Bigrammen verwendet. Die Autoren erzielten in der Online Handschrifterkennung Wort- und Zeichenfehlerraten von 20.3% respektive 11.5% und für den Offline Fall Fehlerraten von 25.9% bzw. 18.2%.

Graves et al. untersuchten in [GS09] die Connectionist Temporal Classification in Verbindung mit einer komplexeren Netzarchitektur, bestehend aus jeweils drei voll-verbundenen Schichten in Abwechslung mit multidimensionalen LSTMs (siehe Kapitel 2.5.4) in einer hierarchischen Anordnung. Die Intention dieser Architektur ist es, komplexere Merkmale schrittweise aufzubauen. Die Autoren wenden ein Box-Verfahren an, bei dem ein Bereich von beispielsweise 4×3 Werten zu einem Vektor mit 12 Elementen zusammengefasst wird. In jedem Schritt betrachtet die LSTM-Schicht eine solche Box und berechnet eine Ausgabe, die wiederum in Boxen aufgeteilt und durch eine voll-verbundene Schicht weiterverarbeitet wird. Die verschiedenen Größen der Eingabebilder erfordern abschließend die Berechnung der Spaltensumme, um die eindimensionale Ausgaberepräsentation für die CTC zu erhalten. Eine Übersicht der Netzarchitektur ist in Abbildung 3.4.1 zu sehen.

Abgesehen von der Zusammenfassung von Bildbereichen bestimmter Größe mithilfe des Box-Verfahrens werden keine heuristisch gewählten Merkmale verwendet. Mit diesem Verfahren konnte mit einer Fehlerrate von 8.57% (Set f) bzw. 21.17% (Set s) eine

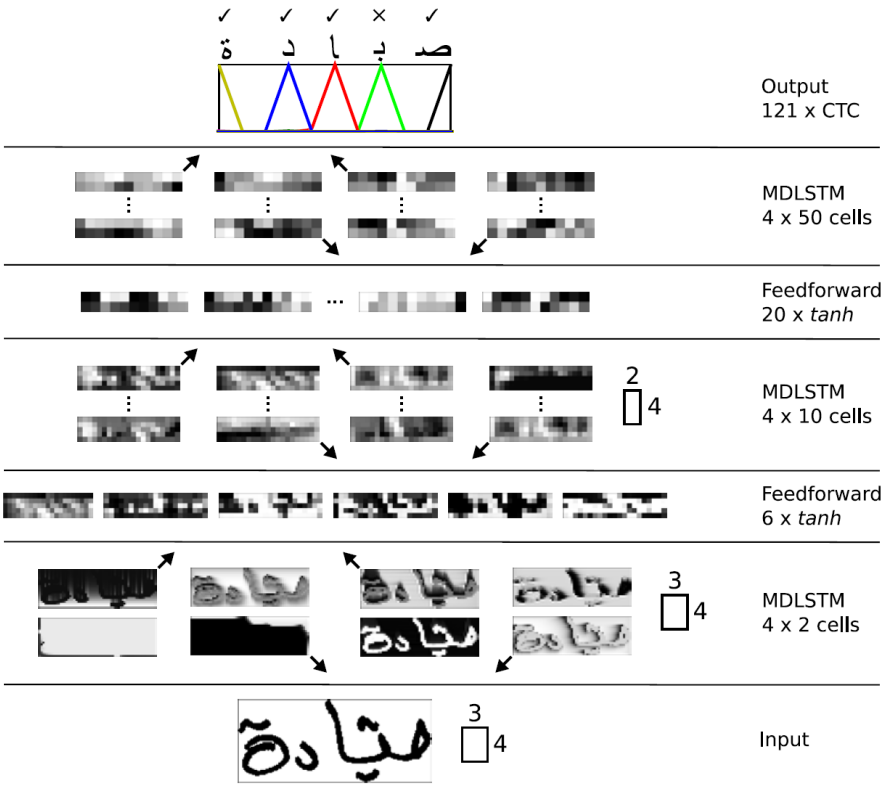


Abbildung 3.4.1: Übersicht über die blockbasierte Connectionist Temporal Classification Architektur aus [GS09]. Über die hierarchische Anordnung von voll-verbundenen Schichten und multidimensionalen LSTMs soll das schrittweise Erlernen komplexer Merkmale begünstigt werden.

deutliche Verbesserung auf dem IFN/ENIT Datensatz [PMM+02] für handschriftlich geschriebene arabische Wörter gegenüber allen konkurrierenden Verfahren des ICDAR 2007 Wettbewerbs [MA07] erzielt werden. Das zweitplatzierte *Siemens-2* Verfahren, das auf Hidden Markov Modelle zur Erkennung setzte, hatte im Vergleich dazu eine Fehlerrate von 12.78% bzw. 26.06%. Im ICDAR 2009 Wettbewerb für Handschrifterkennung [GEA09] führten die Autoren die Rangliste mit dem gleichen Verfahren, aber einer leicht veränderten Parameterkonfiguration an. Sie erzielten auf dem RIMES Datensatz eine Wortfehlerrate von 6.83% bei der Verwendung des kleinen Lexikons mit 1 612 Wörtern und eine Fehlerrate von 8.98% mit dem großen Lexikon von 5 334 Wörtern.

Der Effekt von Dropout in rekurrenten Neuronalen Netzen für die Handschrifterkennung wurde von Pham et al. analysiert [PBKL14]. Sie bewahren die Fähigkeit der Modellierung von Sequenzen der LSTM-Schichten, indem Sie Dropout nur an den ausgehenden und nicht an den rekurrenten Neuronenverbindungen durchführen. Die Autoren experimentierten mit der Anwendung von Dropout an verschiedenen Stellen der MDLSTM Architektur von Graves et al [GS09], in der lediglich die Filtergrößen angepasst wurden. Es konnte gezeigt werden, dass Dropout die Leistungsfähigkeit des rekurrenten Netzes besonders bei einer großen Parameteranzahl verbessert und die Überanpassung reduzieren kann. Für die Einzelworterkennung konnten die Fehlerraten durch die Anwendung von Dropout in der obersten LSTM-Schicht in allen getesteten Fällen auf den RIMES [ACG+06], IAM [LB05] und OpenHaRT [TPMEA14] Datensätzen verringert werden. Auf den RIMES und IAM Datensätzen verbesserte sich die Zeichenfehlerrate des Modells ohne Lexikon durch Dropout von 14.68% auf 12.17% (RIMES) bzw. von 20.07 auf 18.45% (IAM). Bei dem OpenHaRT Datensatz ist mit einer Verbesserung von 12.48% auf 10.97% ein ähnlicher Trend erkennbar. Wird Dropout in allen drei LSTM-Schichten angewendet, ist der erzielte Effekt noch größer. Die Wortfehlerrate konnte so beispielsweise auf dem RIMES Datensatz um weitere 3.55% auf 8.62% gesenkt werden.

3.5 NEURONALE NETZE MIT AUFMERKSAMKEITSMECHANISMUS

Alle bisher vorgestellten Modelle berechnen Merkmale in regelmäßigen Spalten gleicher Größe, horizontal verteilt über das Eingabebild. Dies hat zur Folge, dass Verfahren wie Hidden Markov Modelle oder die Connectionist Temporal Classification nachgeschaltet werden müssen. Die Verarbeitung von mehrzeiligen Texten erfordert daher eine vorherige Zeilen- oder Wortsegmentierung. Auch die Schreibrichtung, welche bei bestimmten Sprachen uneinheitlich sein kann, wird für die jeweilige Architektur stets im Voraus festgelegt. Eine alternative Möglichkeit besteht in der Verwendung Neuronaler Netze mit Aufmerksamkeitsmechanismus (*Neural Networks with Attention*). Ursprünglich entwickelt für die Übersetzung von Texten ermöglicht es der Aufmerksamkeitsmechanismus dem Neuronalen Netz sich in jedem Schritt, abhängig vom Inhalt, auf einen bestimmten Teil der Eingabe zu fokussieren [BCB14].

Die Architektur von Bluche et al. [BLM17] besteht aus den vier Komponenten Encoder, Attention Netzwerk, Zustandsnetzwerk und Decoder. Der Encoder, eine Architektur aus multidimensionalen LSTMs und Faltungsschichten, extrahiert eine Merkmalskarte aus der gesamten Eingabe. Eine weitere multidimensionale LSTM-Schicht bildet das Aufmerksamkeits-Modul, das für jeden Zeitschritt eine Aufmerksamkeits-

karte ausgibt. Die Aufmerksamkeitskarte enthält für alle Stellen der Merkmalskarte einen Wert, der der Aufmerksamkeit entspricht, die im aktuellen Schritt dieser Stelle zukommen soll. Die Aufmerksamkeitskarte ist hierfür Softmax-normalisiert, damit die Summe der Aufmerksamkeit in der gesamten Merkmalskarte über die Zeitschritte hinweg konstant bleibt. Die durch den Encoder berechneten Merkmale werden an jeder Position mit der dazugehörigen Aufmerksamkeit gewichtet und an ein Zustands-LSTM gegeben. Das Zustands-LSTM beobachtet somit den Verlauf der Position der Aufmerksamkeit und den fokussierten Inhalt. Mit den Ausgaben des Zustands-LSTMs bestimmt ein Decoder, bestehend aus einer einzelnen voll-verbundenen Schicht, eine Wahrscheinlichkeitsverteilung über das Alphabet. Ein Blank-Symbol wie bei der Connectionist Temporal Classification ist nicht erforderlich, aber das Ende der Sequenz wird mit einem Sonderzeichen (*End Of Sequence*, EOS) markiert. Eine Übersicht über die verschiedenen Komponenten und ihren Zusammenhang ist in Abbildung 3.5.1 zu finden.

Der Aufmerksamkeits-Mechanismus eröffnet eine Reihe von Möglichkeiten. Durch das schrittweise Fokussieren auf verschiedene Bildbereiche, die anschließend einem Buchstaben zugeordnet werden, können Ausgabesequenzen beliebiger Länge oh-

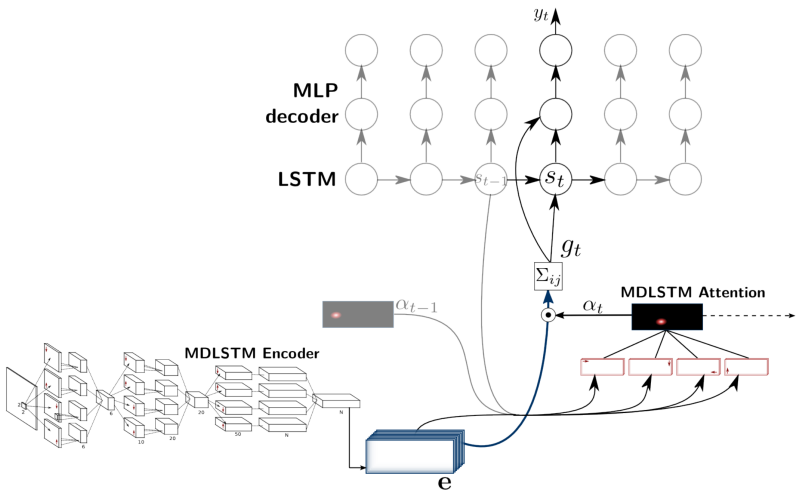


Abbildung 3.5.1: Visualisierung der Aufmerksamkeits-basierten MDLSTM Architektur aus [BLM17]. Ein Attention Netzwerk berechnet auf Basis der durch den Encoder extrahierten Merkmale eine Aufmerksamkeitskarte. Die mit der Aufmerksamkeitskarte gewichteten Merkmale werden durch ein Zustands-LSTM und einen Decoder einem Buchstaben zugeordnet.

ne Nachbearbeitungsschritt aus dem Eingabebild erzeugt werden. Das Attention Netzwerk ermöglicht es außerdem komplexe Leserichtungen abzubilden, was ohne Änderungen der Architektur das Lesen von Wörtern, Zeilen und ganzen Paragraphen ermöglicht. Im Fall von lateinischer Schrift kann es dafür z. B. die hierarchische Leseordnung zunächst von links nach rechts und dann von oben nach unten erlernen.

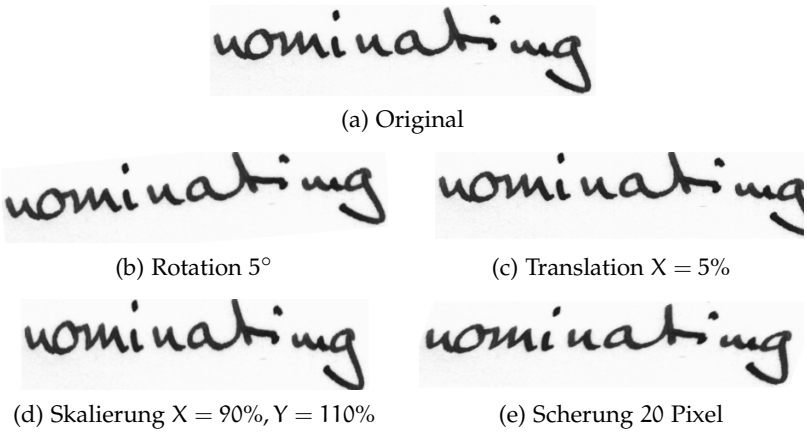
Ein erheblicher Nachteil dieser Methodik ist der extreme Laufzeit- und Speicherplatzbedarf durch die Backpropagation through Time in hunderten von Zeitschritten bei der Transkription ganzer Paragraphen. Die Autoren verkürzten die Berechnung der Backpropagation through Time auf 30 Zeitschritte und wendeten weitere Techniken wie *Curriculum Learning* [BLCW09] an, um das Training des Netzes erfolgreich abschließen zu können. Damit erzielte das Verfahren von Bluche et al. auf dem IAM Datensatz [MB02] eine Zeichenfehlerrate von 7.0% bei der Transkription einzelner Zeilen. Zur Evaluation der Leistungsfähigkeit in der Transkription mehrzeiliger Texte erzeugten die Autoren einen synthetischen Datensatz auf Basis des IAM Datensatzes. Für die Transkription von Zeilenpaaren stieg die Zeichenfehlerrate auf 9.4% an.

3.6 AUGMENTIERUNG IN DER HANDSCHRIFTERKENNUNG

Während Datensätze z. B. für die Szenen Klassifikation wie der Places Datensatz [ZLK+17] oft viele Millionen Bilder umfassen, sind typische Datensätze für die Handschrifterkennung wie die IAM Handwriting Database (ca. 13 000 annotierte Textzeilen und 115 000 annotierte Wörter, siehe Kapitel 5.1.1) um den Faktor 100 bis 1 000 kleiner. Daher ist Augmentierung, wie bereits in Kapitel 2.4.1 erwähnt, besonders in der Handschrifterkennung eine wichtige Technik um den Umfang der Datensätze mit wenig manuellem Aufwand auf realistische Weise künstlich zu vergrößern und die Überanpassung maschineller Lernverfahren zu reduzieren.

Eine verbreitete Variante der Augmentierung in der Bildverarbeitung sind die affinen Transformationen, die in Abbildung 3.6.1 dargestellt sind. Durch Rotation, Translation, Skalierung und Scherung werden die Bilder geringfügig verändert. Im Falle der Objekterkennung oder Segmentierung müssen außerdem die dazugehörigen Annotationen angepasst werden. Krizhevsky et al. verwenden für die Klassifikation von Bildern beispielsweise 224×224 große, zufällig gewählte Bildausschnitte sowie deren horizontale Spiegelungen für das Training ihres tiefen Faltungsnetzes. Diese Augmentierung verhinderte die erhebliche Überanpassung ihres Netzwerks und ermöglichte so erst die Verwendung eines Netzwerks dieser Größe [KSH12].

Simard et al. wenden affine Transformationen in Kombination mit elastischen Deformationen an, die das Bild durch ein Verschiebungsfeld (*displacement field*) manipulieren



(a) Original

(b) Rotation 5°

(c) Translation X = 5%

(d) Skalierung X = 90%, Y = 110%

(e) Scherung 20 Pixel

Abbildung 3.6.1: Anwendung affiner Transformationen auf ein Wortabbild.

[SSP03]. Das Verschiebungsfeld wird dabei zufällig mit Werten zwischen 0 und 1 initialisiert und mit einem Gaussfilter mit Standardabweichung σ geglättet. Anschließend wird das Verschiebungsfeld auf eins normalisiert und mit einem Faktor α skaliert, bevor es auf das Bild angewendet wird. σ und α werden auch als Elastizitäts- und Intensitätskoeffizienten bezeichnet. Einige Beispiele für die elastische Deformation sind in Abbildung 3.6.2 zu sehen.

Ein Problem elastischer Deformationen ist, insbesondere bei einer variablen Größe der Eingabebilder, die Wahl geeigneter Parameter σ und α. Wigington et al. [WSD+17]

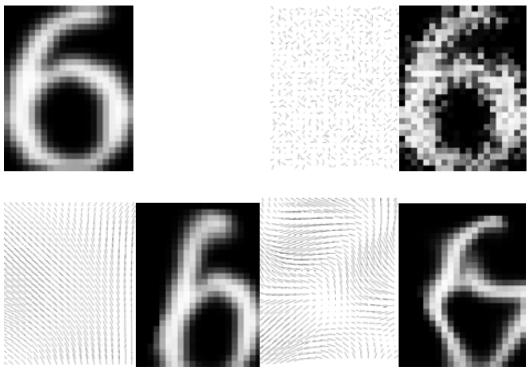


Abbildung 3.6.2: Visualisierung der elastischen Deformation aus [SSP03]. Oben links: Originalbild. Oben rechts und unten: Verschiebungsfelder mit verschiedenen Glättungskoeffizienten und ihre Anwendung auf das Originalbild.

Abbildung 3.6.3: Visualisierung der Random Grid Warp Augmentation von [WSD+17]. Das Bild wird verzerrt, indem ein Gitter über das Bild gelegt wird und die Kontrollpunkte zufällig normalverteilt verschoben werden.

bauen auf den elastischen Deformationen auf und entwickeln das Verfahren durch die Verwendung eines regelmäßigen Gitters weiter, das, ausgerichtet an der Basislinie des Wortes oder der Zeile, über das Bild gelegt wird. Die Kontrollpunkte des Gitters werden anschließend zufällig durch eine Normalverteilung in x und y-Richtung verschoben und das Bild anhand der neuen Positionen der Kontrollpunkte verzerrt. Das *Random Grid Warp Distortion* (RGWD) genannte Verfahren hat mit der Standardabweichung der Normalverteilung σ und dem Abstand der Gitterpunkte d wiederum zwei Parameter. Mit diesen Parametern kann das RGWD Verfahren laut den Autoren zeichenbasierte Variationen in der Handschrift realistisch abbilden und sie sind im Vergleich zu den Elastic Distortions intuitiver einzustellen. In Abbildung 3.6.3 wird das Augmentierungsverfahren auf ein Beispielbild angewendet.

3.7 CLASS ACTIVATION MAPPING

Durch ihre Fähigkeit diskriminative Merkmale für Eingabebilder zu lernen, können Faltungsschichten auch allein für die Objekterkennung eingesetzt werden. Die Möglichkeit Objekte zu lokalisieren geht jedoch durch die nachfolgenden voll-verbundenen Schichten in typischen Faltungsnetzarchitekturen verloren. Werden die voll-verbundenen Schichten durch eine *Global Average Pooling*-Schicht ersetzt, welche die Größe einer Merkmalskarte auf 1×1 reduziert, kann die Lokalisierungsfähigkeit jedoch bis zur Ausgabe des Netzes erhalten werden. Die Bestimmung der diskriminativen Bildbereiche für alle Klassen geschieht dann in einem einzigen Vorwärtsschritt. Zhou et al. [ZKL+16] nennen dieses Verfahren *Class Activation Mapping*. Es ermöglicht eine Lokalisierung von Objekten, ohne Trainingsmaterial zu benötigen, das mit räumlichen Informationen zu den Objekten annotiert ist. Das Netz wird lediglich für die Bildklassifikation und nicht für die Lokalisierung im Bild trainiert, was als *weakly supervised learning* bezeichnet wird. Dabei werden für die jeweiligen Klassen Objektdetektoren gelernt, sodass die Bildinhalte erkannt werden können, die die ausgegebene Wahrscheinlichkeit jeder Klasse bedingen [ZKL+14].

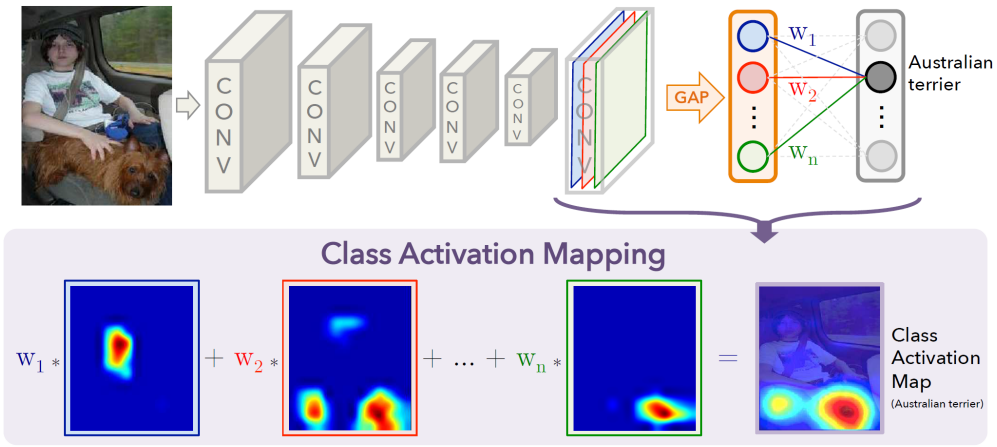


Abbildung 3.7.1: Übersicht über das Class Activation Mapping Verfahren aus [ZKL+16]. Die Heatmap zur Lokalisierung der Objekte wird aus einer gewichteten Summe der Merkmalskarten von den Faltungsschichten gewonnen.

Die Netzarchitektur von Zhou et al. [ZKL+16] besteht aus einer Reihe von Faltungsschichten, gefolgt von einer Global Average Pooling-Schicht. Mit einer finalen voll-verbundenen Schicht wird eine gewichtete Summe über die durch Pooling reduzierten Merkmale gebildet. In Abbildung 3.7.1 ist zu erkennen, wie die voll-verbundene Schicht mit ihren Parametern w_{ij} über Merkmale der Global Average Pooling-Schicht die Multi-Klassenzugehörigkeit bestimmt. Für die Lokalisierung werden die unreduzierten Faltungsmerkmale jeder Position mit den gleichen Parametern w_{ij} gewichtet. Hat Merkmal i bei der Klassifikation auf die Ausgabe für die Klasse j einen Einfluss von w_{ij}, geht auch bei der Lokalisierung der Objektklasse j die i-te Merkmalskarte mit dem Gewicht w_{ij} in die Summe ein. Zur Herstellung der Verbindung zwischen Class Activation Map und den dazugehörigen Bildbereichen müssen die Merkmalskarten auf die Größe des Bildes hoch skaliert werden. Anhand der resultierenden Heatmap jeder Klasse kann die Position und Ausdehnung von Objekten im Bild abgelesen werden. Die Verwendung von Average statt Maximum Pooling soll die Hervorhebung des vollständigen Objektbereichs und nicht nur eines kleinen diskriminativen Teils begünstigen. Die Betrachtung von Bildbereichen, die das Netz für die Klassifikation als entscheidend einstuft, gibt einen Einblick in dessen Funktionsweise und einen Ansatz für die Erklärung eines Erkennungsergebnisses [ZKL+16].

3.8 DISKUSSION

Hidden Markov Modelle haben eine lange, erfolgreiche Historie in der Handschrifterkennung, insbesondere im hybriden Ansatz mit (rekurrenten) Neuronalen Netzen zur Ersetzung heuristisch gewählter Merkmale. Seit der Entwicklung der Connectionist Temporal Classification von Graves et al. finden jedoch Verfahren allein mit Neuronalen Netzen aufgrund ihrer hohen Leistungsfähigkeit vermehrt Verwendung. Besonders in Verbindung mit Long Short-Term Memory Networks zeigt sich die CTC als vielversprechender Ansatz für eine Vielzahl von Aufgaben innerhalb des Sequenzlernens im Allgemeinen und für die Handschrifterkennung im Speziellen. Die Verwendung eines einzelnen Modells, das die Möglichkeit besitzt den gesamten Mustererkennungsprozess Ende-zu-Ende abzubilden, vereinfacht dessen Konzeptionierung. Auch das Training wird erleichtert, da keine komplexen abwechselnden oder integrierten Lernverfahren für die Optimierung von mehreren, voneinander abhängigen Modellen angewendet werden müssen.

Attributraumrepräsentationen sind ein vielversprechender Ansatz für das Wordspotting, benötigen aber für die Transkription stets ein Lexikon, was die Erkennung von Wörtern außerhalb des Lexikons unmöglich macht. Durch den Vergleich mit der Repräsentation aller Elementen des Lexikons im Attributraum, den diese Verfahren für die Zuordnung benötigen, steigt außerdem deren Laufzeit für die Erkennung jedes Wortes mit der Anzahl von Elementen im Lexikon. Netze mit Aufmerksamkeitsmechanismus ermöglichen es, komplexe mehrdimensionale Muster wie eine Textseite mit hierarchischer Leserichtung zu analysieren. Die hohe Laufzeit und der Speicherplatzbedarf beschränken jedoch deren Einsatz in der Praxis. Außerdem sind zum aktuellen Zeitpunkt noch Techniken wie Curriculum Learning und eine Beschränkung der Zeitschritte von der Backpropagation through Time notwendig, um das Training überhaupt erfolgreich abschließen zu können. Die Leistungsfähigkeit auf den typischen Datensätzen zur wort- und zeilenbasierten Transkription befindet sich noch nicht auf dem Niveau des aktuellen Stands der Forschung mit der CTC.

Die Entwicklung aktueller Faltungsnetze, z. B. für die Bildklassifikation, verdeutlicht die Vorteile tiefer Netzarchitekturen, auf die schon die Bezeichnung Deep Learning für diese Kategorie des maschinellen Lernens hinweist. Die hohe Anzahl von Parametern der rekurrenten Netze erschwert die Konzeptionierung leistungsstarker, tiefer Architekturen ausschließlich bestehend aus rekurrenten Schichten. Daher ist die Verbindung von Faltungsschichten für die Merkmalsextraktion und rekurrenten Schichten für das Sequenzlernen ein vielversprechender Ansatz, der in Kapitel 4 vorgestellt wird.

4

METHODIK

Rekurrente Netze wie Long Short-Term Memory Networks wurden, wie im vorhergehenden Kapitel 3 gezeigt, bereits vielfach erfolgreich im hybriden Ansatz oder mit der Connectionist Temporal Classification für verschiedene Aufgaben innerhalb des Sequenzlernens eingesetzt. Ihre Fähigkeit Information über einen längeren Zeitraum zu speichern, ermöglicht es, Muster in ihrem Kontext zu betrachten. Dadurch eignen sie sich besonders für die Verarbeitung von Sequenzen, wie sie z. B. bei der Transkription von Handschrift vorkommen, da Ambiguitäten beispielsweise oftmals durch Informationen aus anderen Sequenzelementen aufgelöst werden können. Der anhaltende Erfolg tiefer Faltungsnetzarchitekturen in vielen Bereichen der Bildverarbeitung, z. B. in der Szenenerkennung [KSH12], Objekterkennung [RHGS15] oder semantischen Segmentierung [LSD15], begründet das Interesse diese auch in der Handschrifterkennung zur Merkmalsextraktion einzusetzen.

Im Folgenden sollen daher die Vorteile der beiden Schichttypen in einer Architektur mit dem Namen *Convolutional Recurrent Neural Networks* (CRNNs) kombiniert werden. Zunächst werden dafür die Convolutional Recurrent Neural Networks eingeführt, bevor auf die Unterschiede der Architekturen von Puigcerver [Pui17] und Wigington et al. [WSD+17] eingegangen wird, die sich grundlegend in dem Aufbau ihrer Faltungs- bzw. rekurrenten Blöcke unterscheiden.

4.1 CONVOLUTIONAL RECURRENT NEURAL NETWORKS

Die Kombination von Faltungsschichten mit rekurrenten Schichten innerhalb einer Netzarchitektur wurde für das bildbasierte Sequenzlernen entwickelt. Dieser Ansatz wird von Shi et al. [SBY17] als Convolutional Recurrent Neural Network (CRNN) bezeichnet. Das folgende Kapitel bezieht sich auf ihre Darstellung. Das CRNN besteht aus drei Bestandteilen: Die Faltungsschichten extrahieren eine Merkmalssequenz aus dem eingegebenen Bild, deren Elemente die rekurrenten Schichten unter Einbezug ihres Kontextes analysieren und eine sogenannte Frame-basierte Vorhersage geben. Den Abschluss bildet eine Transkriptionsschicht, die je Frame die Ausgaben der letzten rekurrenten Schicht in eine Wahrscheinlichkeitsverteilung auf dem Alphabet überführt,

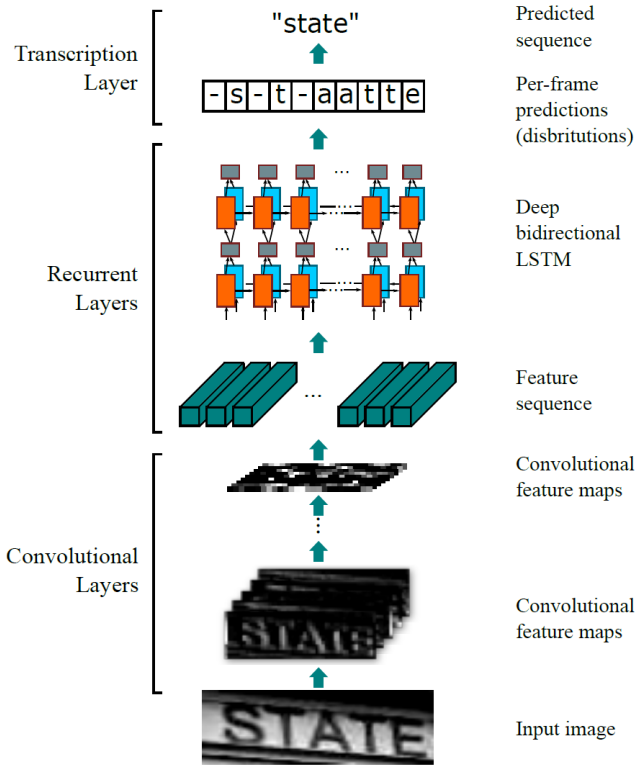


Abbildung 4.1.1: Überblick über die CRNN-Architektur aus [SBY17]. Nach der Berechnung von Merkmalen auf dem Eingabebild durch die Faltungsschichten und die kontextbasierte Analyse durch die rekurrenten Schichten werden die Ausgaben durch eine Transkriptionsschicht in das durch die Connectionist Temporal Classification erforderte Ausgabeformat transformiert.

mit der die korrekte Transkription über die Connectionist Temporal Classification bestimmt werden soll. Diese Struktur wird in Abbildung 4.1.1 dargestellt.

Im Detail entspricht der erste Teil einer einfachen Faltungsnetzarchitektur mit ihren Faltungs- und Pooling-Schichten und unter Ausnahme der voll-verbundenen Schichten. Hierdurch wird eine Merkmalskarte variabler Breite aus dem Eingabebild erzeugt, die von den rekurrenten Schichten weiterverarbeitet wird, indem jede Spalte der Merkmalskarte als Element der Sequenz betrachtet wird. Jedes Element der Eingabe in die erste rekurrente Schicht besteht also aus der Konkatenation aller Ausgabekanäle der letzten Faltungsschicht über die gesamte Höhe der Merkmalskarte. Die Schnittstelle zwischen den Faltungs- und rekurrenten Schichten wird daher auch

als *Map-to-Sequence* Schicht (siehe [SBY17]) oder *Columnwise Concat* (siehe [Pui17]) bezeichnet. Da die rekurrenten Schichten eine feste Dimensionalität der Eingabedaten für jedes Element der Sequenz erwarten, muss die Merkmalskarte der Faltungsschichten eine feste Höhe besitzen. Dies wird erreicht, indem die Eingabedaten innerhalb der Vorverarbeitung auf eine feste Höhe skaliert werden. Ein Element der Sequenz kann somit durch das Rezeptive Feld der Faltungsschichten einer Menge zusammenhängender Spalten im Eingabebild zugeordnet werden und aufeinanderfolgende Elemente entsprechen zusammenhängenden Bereichen der Eingabe, wie in Abbildung 4.1.2 dargestellt ist. Die Faltungs- und Pooling-Operationen erzeugen in Abhängigkeit von der Breite des Höhen-normalisierten Eingabebildes eine Merkmalssequenz variabler Länge.

Die Merkmalskarte der letzten Faltungsschicht wird innerhalb des rekurrenten Teils vom CRNN weiterverarbeitet. Die Verwendung von rekurrenten Schichten, die ebenfalls auf Eingabesequenzen beliebiger Länge operieren können, hat mehrere positive Effekte. Der Einbezug kontextueller Information durch die rekurrenten Schichten ermöglicht die präzisere Bestimmung mehrdeutiger Symbole. Außerdem ist im Fall besonders breit geschriebener Zeichen ein einziges Sequenzelement durch das endlich große rezeptive Feld der Faltungsschichten unter Umständen nicht ausreichend, um das gesamte Zeichen zu umfassen. Im Gegensatz zu Hidden Markov Modellen erlauben rekurrente Netze eine unkomplizierte Fehlerrückführung, was das Ende-zu-Ende Training des gesamten Netzes nach einer zufälligen Initialisierung der Netzparameter ermöglicht. Um die in Kapitel 2.5 angesprochenen Schwierigkeiten bei dem Einsatz allgemeiner rekurrenter Schichten zu umgehen und den Kontext sowohl durch die vorhergehenden als auch folgenden Sequenzelemente zu betrachten, werden

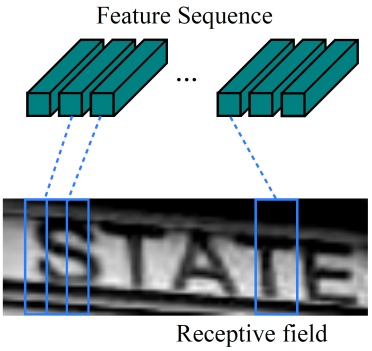


Abbildung 4.1.2: Visualisierung zur Erzeugung der Mermalssequenz für die rekurrenten Schichten aus vollständigen Spalten der Faltungsmerkmale aus [SBY17].

oft bidirektionale LSTMs oder GRUs (siehe Kapitel 2.5.1 und folgende) verwendet. Zur weiteren Steigerung der Leistungsfähigkeit können mehrere solcher Schichten sequentiell angeordnet werden, um ein tiefes rekurrentes Netz zu erhalten.

Die abschließende Schicht, die als Transkriptionsschicht bezeichnet wird, berechnet mithilfe der Connectionist Temporal Classification (siehe Kapitel 2.6) die Transkription aus der Ausgabe der letzten rekurrenten Schicht. Im Normalfall wird zwischen der letzten rekurrenten Schicht und der Transkriptionsschicht zusätzlich eine weitere voll-verbundene Schicht mit einer Softmax-Aktivierungsfunktion eingefügt, da die Connectionist Temporal Classification für jedes Sequenzelement eine Wahrscheinlichkeitsverteilung über das Alphabet L' (inklusive Blank Symbol) erwartet. Die voll-verbundene Schicht überführt die $(2 \cdot D)$-dimensionalen Elemente der Ausgabesequenz von den bidirektionalen rekurrenten Schichten in die $|L'|$-dimensionale Eingabe für die Connectionist Temporal Classification, indem die Berechnung der voll-verbundenen Schicht (mit $(2 \cdot D)$ Eingabeneuronen und $|L'|$ Ausgabeneuronen) auf jedes Element der Sequenz angewendet wird.

4.2 ARCHITEKTUR VON JOAN PUIGCERVER

Joan Puigcerver verwendet für seinen wissenschaftlichen Artikel [Pui17] eine angepasste CRNN-Architektur in Anlehnung an das Netz von Shi et al. [SBY17]. Sie wird in Abbildung 4.2.1 dargestellt. Im Detail kommen dabei fünf Faltungsblöcke zum Einsatz, die jeweils aus einer Faltungsschicht mit steigender Anzahl von Filtern und Dropout, einer Batch Normalisierungsschicht, Rectified Linear Unit Aktivierungsfunktion und Maximum Pooling bestehen. Die Anzahl der Kanäle für die 3×3-Faltungsschicht in Block i beträgt $16 \cdot i$. Padding verhindert dabei die Verkleinerung der Dimensionalität der Ausgabe gegenüber der Eingabe. Dropout wird mit einer Wahrscheinlichkeit von 20% in den letzten drei Blöcken angewendet. Es ist nicht erkennbar, ob es sich dabei um Spatial Dropout handelt oder nicht. Da Trainingsvorgänge mit Spatial Dropout in den hier durchgeführten Experimenten zu konsistenteren Ergebnissen geführt haben, wird für die Experimente in Kapitel 5 Spatial Dropout eingesetzt. Maximum Pooling, jeweils mit Kernelgröße 2×2, wird innerhalb der ersten drei Blöcke angewendet.

Die fünf rekurrenten Blöcke bestehen aus bidirektionalen LSTM-Schichten mit einer konstanten Anzahl von 256 Zellen. Die erste LSTM-Schicht hat eine Eingabedimensionalität, die der Anzahl von Faltungskerneln multipliziert mit der Höhe der Merkmalskarte der letzten Faltungsschicht entspricht. Für die angegebene Architektur der Faltungsblöcke und einer Höhennormalisierung der Eingabebilder auf 128 Pixel ergibt dies eine Höhe der Merkmalskarte von $\frac{128}{8} = 16$ und damit bei 80 Filtern

der letzten Faltungsschicht eine Eingabedimensionalität der ersten LSTM-Schicht von $16 \cdot 80 = 1\,280$. Durch die Bidirektionalität besteht die Ausgabe aller LSTM-Schichten aus $2 \cdot 256 = 512$ Werten pro Sequenzelement. Nach jeder LSTM-Schicht wird Dropout mit einer Wahrscheinlichkeit von 50% angewendet.

Eine voll-verbundene Schicht transformiert die Ausgabe der LSTM Blöcke mit 512 Werten pro Element zu einer Ausgabe mit $|L'|$ Dimensionen bei einem Alphabet der Größe $|L'|$ (inklusive CTC Blank Symbol). Abschließend werden die $|L'|$-dimensionalen Sequenzelemente mit einer Softmax-Schicht in Wahrscheinlichkeitsverteilungen über

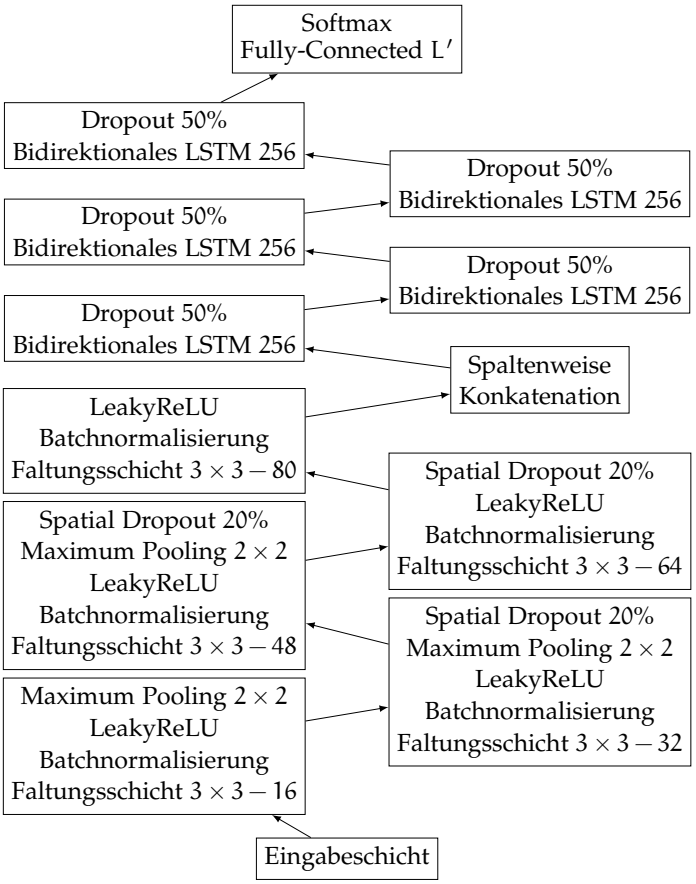


Abbildung 4.2.1: Visualisierung der CRNN-Architektur von Joan Puigcerver [Pui17]. Aus dem Eingabebild werden mit fünf Faltungsschichten Merkmale extrahiert, die anschließend durch fünf bidirektionale LSTM-Schichten weiterverarbeitet werden. Eine voll-verbundene Schicht bringt die Ausgabe in das richtige Format für die Connectionist Temporal Classification.

das Alphabet transformiert, die durch die Connectionist Temporal Classification verarbeitet werden können. Die Architektur besitzt abhängig von der Höhennormalisierung der Eingabebilder und dem Alphabet insgesamt ca. 9.6 Millionen Parameter. Dabei entfallen ca. 98% der Parameter (9.5 Millionen) auf die rekurrenten Schichten.

Puigcerver rechtfertigt die Verwendung der CRNN-Architektur gegenüber den verbreiteten multidimensionalen LSTM Netzen durch deren Berechnungskomplexität und die visuelle Ähnlichkeit der auf den unteren Schichten berechneten Merkmale [Pui17]. Während eine parallele Implementierung einer 2D-LSTM-Schicht für ein Bild der Größe $H \times W$ mit C Ein- und D Ausgabekanälen eine Laufzeit von $\mathcal{O}((W + H) \cdot D + C)$ hat, kann eine Faltungsschicht mit einem Rezeptiven Feld mit S Elementen in Laufzeit $\mathcal{O}(C \cdot S)$ berechnet werden. Besonders in den unteren Schichten des Netzes kann $\mathcal{O}((W + H) \cdot D + C)$ um den Faktor 10 oder 100 größer sein als $\mathcal{O}(C \cdot S)$. Zusammengenommen mit der visuellen Ähnlichkeit der durch die beiden Schichttypen berechneten Merkmale, begründet dies die Entscheidung Puigcervers auf multidimensionale LSTM-Schichten zu verzichten und Faltungsschichten zur Extraktion der Merkmale zu verwenden. In Abbildung 4.2.2 wird eine zufällige Auswahl von Merkmalen einer 2D-Schicht und einer Faltungsschicht zum qualitativen Vergleich gegenübergestellt. In den hinteren Schichten der Netzarchitektur genügt nach Puigcerver die Verwendung eindimensionaler LSTMs zur Modellierung der sprachlichen Eigenschaften, da Sprache im Allgemeinen durch eindimensionale Zeichensequenzen (Wörter und Sätze) ausgedrückt werden kann.

(a) 2D-LSTM-Schicht

(b) Faltungsschicht

Abbildung 4.2.2: Gegenüberstellung zufälliger Merkmale einer 2D-LSTM und Faltungsschicht an vergleichbaren Stellen der Architekturen aus [Pui17].

4.3 ARCHITEKTUR VON WIGINGTON ET AL.

Auch Wigington et al. setzen zur Vorstellung ihres Datenaugmentierungsverfahrens für die Handschrifterkennung ein Convolutional Recurrent Neural Network [WSD+17] ein. Die hier verwendete Implementierung, die in Abbildung 4.3.1 visualisiert wird, setzt mit sechs 3×3 Faltungsschichten, bestehend aus 64, 128, 256, 256, 512 und 512 Kanälen, deutlich mehr Parameter in den Faltungsblöcken ein. Batchnormalisierung wird nur nach den ersten zwei Faltungsschichten und 2×2 Maximum Pooling nach der ersten, zweiten, vierten und sechsten Schicht angewendet. Eine Besonderheit besteht in der Schrittweite der Maximum Pooling-Schicht. Für die Merkmale der ersten zwei Faltungsschichten wird ein nicht überlappendes Maximum Pooling mit einer Schrittweite von zwei in beiden Richtungen eingesetzt, während die horizontale Schrittweite in den letzten beiden Maximum Pooling-Schichten auf eins reduziert wird. Nach dem *Columnwise Concat* folgen zwei bidirektionale LSTM-Schichten mit 512 bzw. 256 Zellen nach jeweils einer Dropout Schicht mit 50% Wahrscheinlichkeit. Die Transkriptionsschicht überführt die 512-dimensionale Ausgabe der letzten

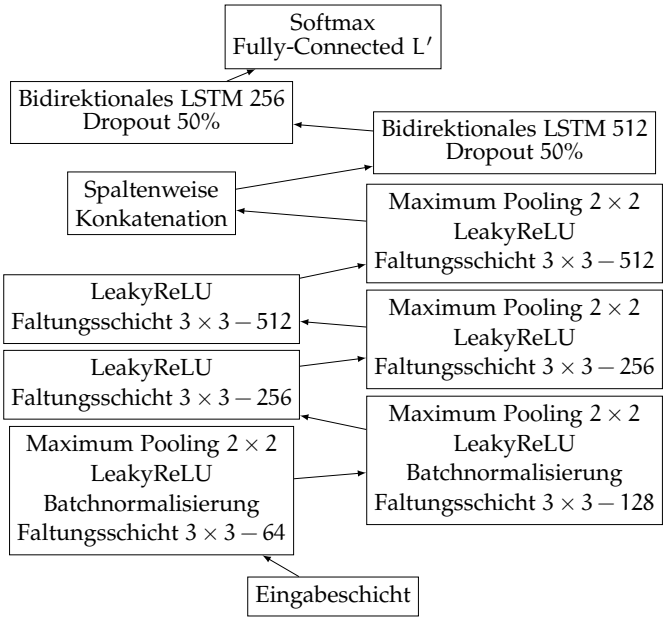


Abbildung 4.3.1: Visualisierung der CRNN-Architektur von Wigington et al. [WSD+17]. Sechs Faltungsschichten extrahieren Merkmale aus dem Eingabebild, die durch zwei bidirektionale LSTM-Schichten weiterverarbeitet werden, bevor eine voll-verbundene Schicht die Ausgabe in das korrekte Format überführt.

LSTM-Schicht in L′-dimensionale Sequenzelemente, bevor die Softmax-Schicht die letztendliche Wahrscheinlichkeitsverteilung für die Weiterverarbeitung innerhalb der CTC berechnet.

Mit ca. 19.8 Millionen Parametern ist die Architektur von Wigington et al. im Vergleich zur Architektur von Puigcerver etwa doppelt so groß. Dies ist vor allem auf die größere Anzahl an Eingabewerten in die LSTM-Schichten zurückzuführen, da die letzte Faltungsschicht 512 statt 80 Merkmalskarten berechnet. Der Anteil der Parameter in den rekurrenten Schichten liegt durch den größeren Umfang der Faltungsschichten mit 15.3 Millionen bei ca. 77%.

4.4 CLASS ACTIVATION MAPS FÜR CRNNS

Um einen Einblick in die Arbeitsweise und die Funktion der Faltungs- und rekurrenten Blöcke der CRNN-Architektur zu erhalten, ist die Betrachtung der von den Faltungsblöcken berechneten Information wertvoll. Zu diesem Zweck können Class Activation Maps (siehe Kapitel 3.7) eingesetzt werden, die die Aktivierungen für die Erkennung einzelner Buchstaben eines Eingabebildes sichtbar machen. Dafür wird eine Global Average Pooling-Schicht und eine voll-verbundene Schicht für das Class Activation Mapping auf Basis der durch die Faltungsschichten berechneten Merkmale in die Architektur eingefügt. Das sich daraus ergebene Schema wird in Abbildung 4.4.1 dargestellt. Nach dem Training des CRNNs für die Transkription mit der CTC kann die voll-verbundene Schicht für das Class Activation Mapping auf die Erkennung der Anwesenheit der einzelnen Zeichen des Alphabets trainiert werden, während die Gewichte des restlichen Netzes unverändert bleiben. Mit den so angepassten Parametern für die voll-verbundene Schicht der Class Activation Architektur kann daraufhin über die Heatmaps (siehe Kapitel 3.7) die Position der Zeichen im Eingabebild bestimmt werden. Da die Class Activation Architektur direkt mit den Merkmalen der Faltungsschichten arbeitet, die auch dem rekurrenten Teil der CRNN-Architektur als Eingabe dienen, kann so das Zusammenspiel der beiden Teil-Netze analysiert werden.

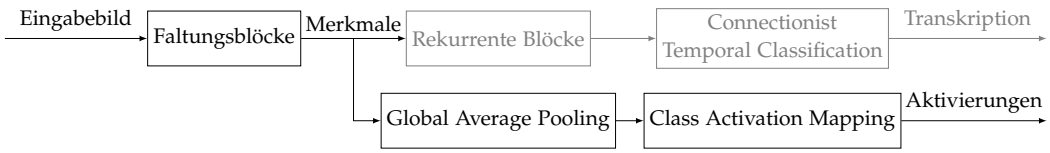


Abbildung 4.4.1: Schema des Class Activation Mappings für die Handschrifterkennung auf der Basis von Convolutional Recurrent Neural Networks.

5 EVALUATION

Zur Evaluation der Methodik aus Kapitel 4 enthält das folgende Kapitel verschiedene Experimente. In Abschnitt 5.1 werden zunächst die Datensätze und Metriken zur Bewertung der Leistungsfähigkeit der verschiedenen Architekturen vorgestellt. Abschnitt 5.3 enthält einige Details zum Experimentalsetup und dessen Implementierung. Die Ergebnisse für den Vergleich mit dem State-of-the-Art werden in Abschnitt 5.4 vorgestellt. Hier wird auch der Einfluss von der Wortlänge, der verschiedenen Architekturentscheidungen und der Häufigkeit der Zeichen im Trainingsdatensatz auf die Transkriptionsleistung betrachtet. Je nach Datensatz wird sowohl die wort- als auch zeilenbasierte Transkription ausgewertet.

5.1 DATENSÄTZE

Für die Evaluation der Verfahren werden Stichproben benötigt, die Paare von Scans von Wörtern oder Zeilen mit ihren zugehörigen Transkriptionen enthalten. Um eine Beurteilung verschiedener Schriftstile durchführen zu können und den Einfluss von akzentuierten Buchstaben zu analysieren, wird eine Auswahl von Datensätzen mit modernen und historischen Dokumenten in verschiedenen Sprachen betrachtet. Diese werden in den folgenden Abschnitten beschrieben.

5.1.1 *IAM Handwriting Database*

Die IAM Handwriting Database [MB02] ist eine der umfangreichsten Datensätze für die Evaluation von Systemen zur Transkription von uneingeschränkter Handschrift. Die Version 3.0 umfasst Abbilder von 1 539 Seiten Text geschrieben von 657 Personen und enthält insgesamt 115 320 Wörtern in 13 353 Zeilen. Ein Beispiel für eine geschriebene Textzeile ist in Abbildung 5.1.1 zu sehen.

Nach einer Registrierung können alle erforderlichen Dateien von der Website[1] der Research Group on Computer Vision and Artificial Intelligence der Universität Bern heruntergeladen werden. Es existiert eine Einteilung in Trainings-, Validierungs- und Test-Sets, die jeweils 6 161, 1 840 und 1 861 Zeilen bzw. 53 839, 16 465 und 17 616 Wörter

[1] http://www.fki.inf.unibe.ch/databases/iam-handwriting-database

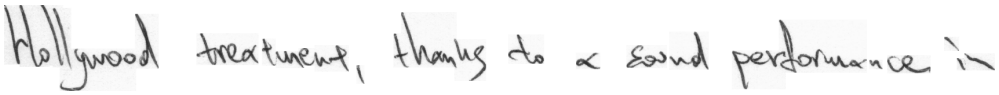


Abbildung 5.1.1: Beispiel für eine Zeile aus der IAM Handwriting Database [MB02].

beinhalten. 3 491 Zeilen und 27 400 Wörter wurden somit keinem Set zugeordnet. Die Beispiele sind mit der dazugehörigen Identifikationsnummer der Seite, Koordinaten der Bounding-Box, der dazugehörigen Transkription und einem Fehlerindikator annotiert. Der Indikator gibt dabei an, ob das automatisierte Segmentierungsverfahren bei der Erzeugung des Datensatzes für dieses Beispiel korrekt funktioniert hat. Alle Beispiele, die unter Umständen nicht korrekt segmentiert wurden, die eine ungültige Bounding-Box haben oder deren Bilddatei nicht lesbar ist, wurden verworfen. Deshalb ergeben sich für die hier durchgeführten Experimente die in Tabelle 5.1.1 angegebenen Größen der verschiedenen Sets. Die Trainings-, Validierungs- und Test-Sets wurden so aufgeteilt, dass jeder Schreiber nur zu Beispielen eines Sets beigetragen hat. Damit sind den Verfahren die Handschriften der Schreiber des Validierungs- und Test-Sets auch nach dem Training noch unbekannt. Die Annotation umfasst insgesamt 79 verschiedene Zeichen. Um einen möglichst fairen Vergleich mit anderen Verfahren zu ermöglichen, wurden die Netze, sofern nicht anders angegeben, für die Transkription des gesamten Zeichensatzes trainiert.

5.1.2 *RIMES Database*

Ein weiterer verbreiteter Datensatz für die Handschrifterkennung ist die RIMES Database [ACG+06]. In Abbildung 5.1.2 ist beispielhaft eine Zeile aus dem Datensatz abgebildet, der von der A2IA (Artificial Intelligence and Image Analysis) Gruppe zusammengestellt wurde. Insgesamt umfasst die RIMES Database 5 605 Formulare

Größe der Sets des IAM Datensatzes				
	Training	Validierung	Test	Gesamt
Zeilen	5 419	747	1 480	7 646
Wörter	46 762	6 414	13 644	66 820

Tabelle 5.1.1: Größe der Sets des IAM Handwriting Datensatzes nach dem Verwerfen ungültiger Beispiele.

Abbildung 5.1.2: Beispiel für eine Zeile aus der RIMES Database [ACG+06].

mit 12 723 Seiten Text von über 1 300 Freiwilligen. Den Personen wurde eine fiktionale Identität und bis zu fünf Szenarien vorgeschlagen, zu denen Sie eine Postnachricht in französischer Sprache verfassen sollten. Auch beim RIMES Datensatz gibt es eine Einteilung in Trainings-, Validierungs- und Test-Sets. Für den zeilenbasierten Fall wurde das Test-Set jedoch nicht veröffentlicht. Nach der Entfernung von Einträgen mit leerer Annotation ergibt sich eine Gesamtanzahl von 7 646 Zeilen und 66 820 Wörtern im Datensatz. Die Größe der einzelnen Sets wird in der Tabelle 5.1.2 angegeben. Wie bei der IAM Handwriting Database trägt jeder Autor nur zu einem Set Beispiele bei.

Die Datenbank kann nach dem Ausfüllen einer Nutzungsvereinbarung von der A2IA Website[2] heruntergeladen werden. Die Annotation enthält zur Berücksichtigung zusätzlicher Sonderzeichen und vor allem Buchstaben mit Akzenten insgesamt 99 verschiedene Zeichen. Wie bei der IAM Database wurden die Netze auf Basis des gesamten Zeichensatzes trainiert, um die Vergleichbarkeit mit möglichst vielen Verfahren sicherstellen zu können.

5.1.3 *George Washington Database*

Während die IAM und RIMES Datensätze Beispiele moderner Handschriften enthalten, basiert die George Washington Database auf Teilen der George Washington Papers aus dem 18. Jahrhundert [FKFB12]. Sie wurde, wie die IAM Database, von der Computer Vision and Artificial Intelligence Forschungsgruppe der Universität Bern erstellt und

Größe der Sets des RIMES Datensatzes

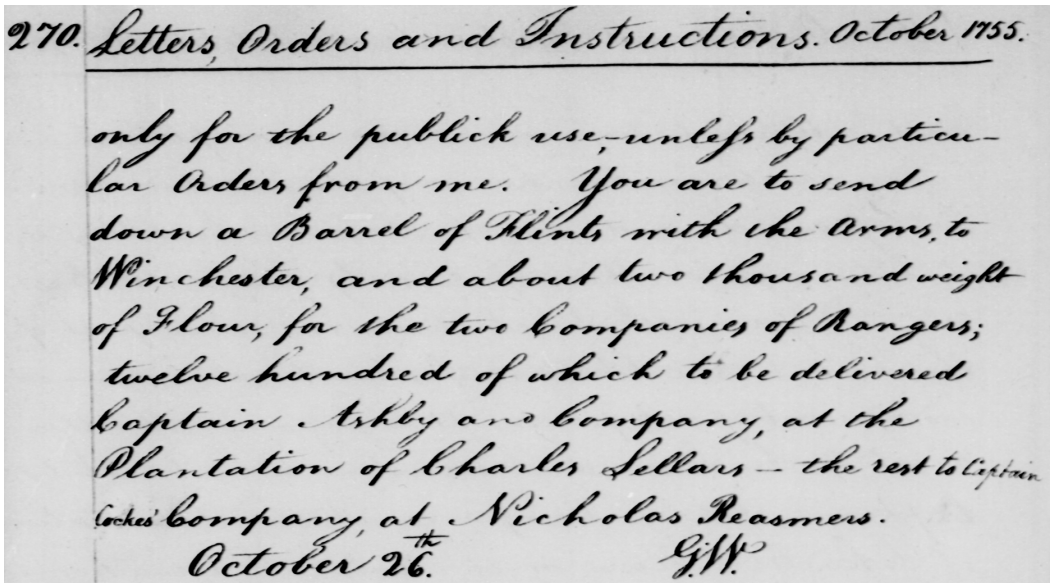
	Training	Validierung	Test	Gesamt
Zeilen	10 733	746	–	11 479 + |Test|
Wörter	51 739	7 464	7 776	66 979

Tabelle 5.1.2: Größe der Sets des RIMES Datensatzes nach dem Verwerfen ungültiger Beispiele.

[2] http://www.a2ialab.com/doku.php?id=rimes_database:start

kann ebenfalls nach einer Registrierung auf ihrer Website[3] heruntergeladen werden. Im Gegensatz zu den anderen Datensätzen ist die ausgeprägt kursive Handschrift sehr einheitlich, da es sich bei den Dokumenten um eine Abschrift von Briefen George Washingtons durch einen einzelnen Schreiber handelt. Ein Beispiel für einen Textparagrafen ist in Abbildung 5.1.3 zu sehen.

Die George Washington Database ist mit einem Umfang von 4 860 Wörtern auf 20 Seiten der kleinste hier verwendete Datensatz. Die vorliegende Form des Datensatzes enthält keine zeilenbasierte Annotation. Eine offizielle Einteilung in Trainings-, Validierungs- und Test-Sets existiert auch nicht. Aus diesem Grund wurden für die Experimente in Kapitel 5.4 10% der Wortbeispiele aus dem Training für die Evaluation zurückgehalten. Auf eine Kreuzvalidierung für die Ermittlung der Leistungsfähigkeit der Verfahren wurde aufgrund des erheblichen Aufwands für das mehrfache Training der Netze verzichtet. Die vorliegende wortbasierte Annotation der George Washington Database wurde bereits von Sonderzeichen befreit und enthält nur die Zeichen von *a* bis *z* und *o* bis *9*.


270. Letters, Orders and Instructions. October 1755.

only for the publick use, unless by particular Orders from me. You are to send down a Barrel of Flints with the Arms, to Winchester, and about two thousand weight of Flour, for the two Companies of Rangers; twelve hundred of which to be delivered Captain Ashby and Company, at the Plantation of Charles Sellars — the rest to Captain Cockes Company, at Nicholas Reasmers.

October 26th G.W.

Abbildung 5.1.3: Ein Auszug aus der George Washington Database [FKFB12].

[3] http://www.fki.inf.unibe.ch/databases/iam-historical-document-database/washington-database

5.2 METRIKEN

Zur Bewertung der Leistungsfähigkeit der Methodik werden Metriken benötigt, mit denen eine Aussage über die Qualität des Transkriptionsergebnisses getroffen werden kann. Anders als bei typischen Klassifikationsaufgaben ist eine einfache Fehlerrate, die den Anteil korrekt klassifizierter Beispiele angibt, nicht ausreichend. Da es sich bei der Ausgabe um Sequenzen handelt, können lediglich einzelne Sequenzelemente gegenüber der korrekten Transkription durch ein fehlerbehaftetes Verfahren zu wenig oder zu viel enthalten oder ersetzt worden sein. Es muss also zwischen Sequenzen mit wenigen und vielen Fehlern unterschieden werden. Aus diesem Grund haben sich für die Beurteilung innerhalb der Schrifterkennung Metriken auf Basis der Levenshtein Editierdistanz etabliert [Lev66].

ZEICHEN- UND WORTFEHLERRATE Die Levenshtein Distanz zwischen zwei Sequenzen, der Hypothese und der Referenz, wird definiert über die minimale Anzahl von Einfügungen, Entfernungen und Ersetzungen von Elementen der Hypothese, um die Referenz zu erhalten. Sie kann über dynamische Programmierung bestimmt werden. Die Zeichenfehlerrate (*Character Error Rate*, CER) und Wortfehlerrate (*Word Error Rate*, WER) entsprechen den auf die Länge der Referenz normalisierten Levenshtein Distanzen und können durch

$$\mathrm{WER} = \frac{w_{\mathrm{sub}} + w_{\mathrm{ins}} + w_{\mathrm{del}}}{w_{\mathrm{ref}}} \tag{5.2.1}$$

$$\mathrm{CER} = \frac{c_{\mathrm{sub}} + c_{\mathrm{ins}} + c_{\mathrm{del}}}{c_{\mathrm{ref}}} \tag{5.2.2}$$

berechnet werden. Die Werte w_{sub}, w_{ins} und w_{del} (c_{sub}, c_{ins} und c_{del}) beziehen sich dabei auf die Anzahl an erforderlichen Ersetzungen, Einfügungen und Entfernungen von Wörtern (Zeichen). w_{ref} (c_{ref}) ist die Anzahl an Wörtern (Zeichen) in der Referenz. Auch wenn diese Fehlermaße meist als prozentuale Werte angegeben werden, sind durch die Berücksichtigung der Einfügungen mehr als 100% Fehlerrate möglich [Blu15, S. 54]. Im Normalfall ist die Wortfehlerrate bei der Beurteilung von zeilenbasierten Transkriptionsverfahren größer als die Zeichenfehlerrate, da bereits ein einzelner Fehler in einem Wort zur Ersetzung des gesamten Wortes bei ihrer Berechnung führt. Beispielsweise beträgt die Wortfehlerrate einer Zeile mit 12 Wörtern und 60 Zeichen bei jeweils einer Buchstabenvertauschung in zwei Wörtern $16.\overline{6}\%$, während die Zeichenfehlerrate nur bei $3.\overline{3}\%$ liegt. Bei der wortbasierten Transkription entspricht die Wortfehlerrate dem Anteil falsch erkannter Wörter.

STATISTISCHES TESTEN Die Berechnung einer Metrik auf einem Validierungs- oder Test-Set entspricht einer Schätzung für die Leistungsfähigkeit eines Verfahrens. Um eine Auskunft darüber geben zu können, wie sicher die Schätzung der Leistungsfähigkeit über die Metrik anhand dieser Stichprobe ist, können Konfidenzintervalle angegeben werden. Ein Konfidenzintervall entspricht dabei einem Bereich von Werten, der das korrekte Ergebnis (die echte Zeichen- oder Wortfehlerrate des Verfahrens) mit einer vorgegebenen Wahrscheinlichkeit enthält. Als Wahrscheinlichkeit, die man als Konfidenzniveau bezeichnet, wird oft 95% gewählt. Sind die Daten z. B. normalverteilt und die Standardabweichung σ bekannt, kann das 95%-Konfidenzintervall für den Mittelwert μ durch

$$\left[\bar{x} - 1.96\frac{\sigma}{\sqrt{n}}, \bar{x} + 1.96\frac{\sigma}{\sqrt{n}}\right]$$

berechnet werden, wenn der empirische Mittelwert der Daten $\bar{x}$ entspricht und die Stichprobe n Elemente umfasst [DKLM05, S. 346f].

Für die Zeichen- und Wortfehlerraten sind jedoch die zugrundeliegenden Verteilungen der Daten unbekannt, sodass die Konfidenzintervalle nicht auf diese Weise berechnet werden können. Stattdessen werden sogenannte empirische Bootstrap Konfidenzintervalle eingesetzt, die eine Intervallschätzung über ein vielfaches probabilistisches Ziehen mit Zurücklegen aus der Stichprobe ermöglicht. Das folgende Vorgehen orientiert sich an der Darstellung in [DKLM05, S. 269ff] von Dekking et al.

Formal werden die Daten $x_1, x_2, \ldots, x_n$ zur Berechnung der Bootstrap Konfidenzintervalle als Realisierung der Stichprobe $X_1, X_2, \ldots, X_n$ der Verteilung F betrachtet. Sei h eine Statistik (z. B. die Zeichen- oder Wortfehlerrate), die auf der Stichprobe berechnet wird. Wird die Realisierung der Stichprobe $x_1, x_2, \ldots, x_n$ durch $x_1^*, x_2^*, \ldots, x_n^*$, also eine Realisierung gleicher Größe durch Ziehen mit Zurücklegen ersetzt, ist die sich ergebende Verteilung $\hat{F}$ eine Approximation von F. Damit wird auch die Verteilung von $h(X_1, X_2, \ldots, X_n)$ durch $h\left(X_1^*, X_2^*, \ldots, X_n^*\right)$ approximiert.

Konkret wird für die Berechnung des 95% Konfidenzintervalls der Zeichenfehlerrate eine Menge (z. B. 10 000) von Bootstrap Realisierungen durch Ziehen mit Zurücklegen aus den Beispielen des Validierungs-Datensatzes gezogen. Im Anschluss wird die Zeichenfehlerrate jeder Bootstrap Realisierung berechnet und die Fehlerrate des Validierungs-Sets abgezogen, um die jeweilige Abweichung zu erhalten. Die Liste der 10 000 Abweichungen der Bootstrap Zeichenfehlerrate von der empirischen Zeichenfehlerrate wird sortiert und das 2.5-Perzentil sowie das 97.5-Perzentil berechnet. Die Summe des jeweiligen Perzentils mit der empirische Fehlerrate ergibt die untere bzw. obere Grenze für das Konfidenzintervall der Zeichenfehlerrate, in dem 95% der Wahrscheinlichkeitsmasse enthalten sind.

Trotz überlappender Konfidenzintervalle kann der Leistungsunterschied zweier Verfahren jedoch statistisch signifikant sein. Um dies zu überprüfen, eignen sich statistische Signifikanztest. Bei einem Signifikanztest wird eine Nullhypothese aufgestellt, die z. B. besagt, dass die Zeichenfehlerraten zweier Verfahren gleich sind $H_0 : CER_1 = CER_2$. Daraufhin kann überprüft werden, ob H_0 anhand der vorliegenden Daten zu einem bestimmten Testniveau (z. B. $\alpha = 5\%$) abgeleht werden kann. Ist das der Fall, führt das zur Annahme der alternativen Hypothese $H_A : CER_1 \neq CER_2$.

Auch bei den statistischen Tests existiert das Problem der Unbekanntheit der zugrundeliegenden Verteilungen. Daher sollten Verfahren wie Randomisierungs-Tests (auch Permutierungs-Test genannt) oder die Bootstrap Shift Methode eingesetzt werden, die keine Annahme über die zugrundeliegende Verteilung der Daten machen. Der folgende Abschnitt zum Randomisierungs-Test folgt der Darstellung in [SAC07], angepasst für die Berechnung einer Zeichenfehlerrate.

Die grundlegende Idee des Randomisierungs-Test ist die Annahme, dass die Daten aus der gleichen Verteilung stammen und die Zuordnung zu einem der beiden Verfahren zufällig getroffen worden ist. Jeder Datenpunkt (Anzahl Zeichen des Test-Beispiels und Anzahl von Zeichenfehlern) hat nach diesem Verständnis zufällig ein Label A oder B erhalten, sodass auch die berechnete Leistungsdifferenz $CER_{diff} = |CER_1 - CER_2|$ zufällig entstanden ist. Somit hat jede der insgesamt N_{perm} Zuordnungen (Permutationen der Labels) das Potenzial, eine größere zufällige Leistungsdifferenz zu haben als die ursprünglich berechnete CER_{diff}. Existieren jedoch weniger als $\alpha \cdot N_{perm}$ Zuordnungen mit einer Leistungsdifferenz größer als CER_{diff}, so haben die Verfahren einen signifikanten Leistungsunterschied und H_0 kann abgelehnt werden. Dies gilt analog für andere Metriken wie die Wortfehlerrate. Da die Anzahl an Permutationen exponentiell mit der Größe des Datensatzes wächst, werden nicht alle, sondern meist nur eine ausreichend große Anzahl zufälliger Permutationen (z. B. $N_{perm} = 100\,000$) betrachtet.

5.3 IMPLEMENTIERUNG

Für die Entwicklung der Trainings- und Evaluationsmethoden sowie den Augmentierungstechniken wird die Programmiersprache Python[4] in der Version 3.7 verwendet. Die Netzarchitekturen werden mithilfe des Deep Learning Frameworks PyTorch[5] 1.0.1 implementiert, das neben vielen vorgefertigten Schichttypen eine einfache Schnittstelle für die Beschleunigung der Berechnungsvorgänge mit CUDA in Verbindung mit

[4] https://www.python.org/
[5] https://pytorch.org/

Nvidia Grafikkarten bietet. Zentraler Datenspeicher und Basis für alle Operationen sind die sogenannten Tensoren, Datenstrukturen, die für Bilddaten meist die Form (Batchgröße × Kanäle × Höhe × Breite) haben. Tensoren können zwischen dem Arbeitsspeicher und dem Grafikspeicher für die GPU-Beschleunigung verschoben werden. Alle Trainingsvorgänge werden auf einer Nvidia GeForce GTX Titan X Grafikkarte mit 12 GB Speicher durchgeführt. PyTorch unterstützt die automatische Berechnung von Gradienten mit dem autograd-Paket [PGC+17], sodass jeweils nur der Vorwärtsschritt durch die Netzarchitekturen implementiert werden muss.

Der Optimierungsprozess wird mit dem RMSProp Verfahren mit einer Batchgröße von 16 durchgeführt (siehe Kapitel 2.2.2). Jede Trainingsepoche enthält alle Trainingsbeispiele in zufälliger Reihenfolge, die vor der Verarbeitung durch das Neuronale Netz optional einmalig durch das ausgewählte Augmentierungsverfahren verändert werden. Abhängig von der Netzarchitektur ist die Lernrate für den Trainingsprozess entweder $3 \cdot 10^{-4}$ (Puigcerver) oder $5 \cdot 10^{-5}$ (Wigington et al.). Für beide Architekturen werden Momentum und Alpha auf 0.5 bzw. 0.95 festgelegt. Um den Trainingsprozess erfolgreich abzuschließen, müssen die Gradienten für die Architektur von Wigington et al. teilweise beschnitten werden. Beim sogenannten Gradient Clamping wird dafür gesorgt, dass der Betrag der Gradienten innerhalb der Backpropagation einen festgelegten Wert (beispielsweise 100) nicht übersteigt. Eine weitere Besonderheit betrifft die Verwendung von Minibatch-basiertem Gradientenabstieg in Kombination mit rekurrenten Schichten. Enthält der Minibatch Elemente mit verschiedenen zeitlichen Ausdehnungen, müssen alle Elemente durch Padding auf die gleiche Länge gebracht werden. Damit die rekurrenten Schichten die so hinzugefügten Werte bei der Berechnung nicht berücksichtigen, existiert in PyTorch die spezielle *PackedSequence* Container-Datenstruktur, die die Eingabedaten kapselt.

Für die Augmentierung der Trainingsbeispiele wird die *affine* Methode des Python-Pakets imgaug[6] und der Random Grid Warp Distortion Algorithmus[7] von Curtis Wigington verwendet. Die Parameter für die affine Augmentierung orientieren sich an der Implementation von Joan Puigcerver und entsprechen jeweils den Grenzen für die zufallsbasierte Ziehung aus einer Gleichverteilung. Der Bildinhalt wird sowohl in horizontaler als auch in vertikaler Richtung unabhängig voneinander um bis zu 5% gestaucht oder gestreckt und um bis zu 5° mit oder gegen den Uhrzeigersinn rotiert. Über den Translationsparameter werden die Bildinhalte in horizontaler und vertikaler Richtung um bis zu 5% verschoben und Scherung wird mit bis zu 5 Pixeln angewendet. Im Falle der zeilenbasierten Transkription werden die Parameter für die

[6] https://imgaug.readthedocs.io/
[7] https://github.com/cwig/handwriting_augmentation

Rotation auf 2.5° und für die Translation in horizontaler Richtung auf 2% reduziert, um den Umfang der Bildinhalte, die aus dem Bildbereich verloren gehen, gering zu halten. Die Füllfarbe für neue Pixel, die durch die affinen Augmentierungsoperationen hinzukommen, ist entsprechend des Texthintergrunds weiß. Die Random Grid Warp Distortion Augmentierung wird für alle Experimente mit ihren Standardparametern von 25 Pixeln Abstand der Gitterpunkte und einer Standardabweichung von 3 für die zufallsbasierte Verschiebung der Gitterpunkte angewendet. Wigington et al. führen vor der Anwendung der RGWD-Augmentierung eine Profilnormalisierung durch und passen die Gittergröße und Position an die Buchstabengröße und -position an, um die Erzeugung von unnatürlichen Knicken durch das Augmentierungsverfahren zu vermeiden. Da nicht für alle Datensätze Informationen über den Masseschwerpunkt und die Basislinie der Schrift verfügbar sind, wird dies in den folgenden Experimenten vernachlässigt. Alle Experimente wurden mit dem Best-Path Dekodierungsverfahren durchgeführt.

5.4 ERGEBNISSE

Dieser Abschnitt enthält die Ergebnisse der Experimente zu den in Kapitel 4 vorgestellten Netzen von Puigcerver und Wigington et al. Die beiden CRNN-Architekturen werden dabei bezüglich ihrer Leistungsfähigkeit verglichen und es wird der Effekt verschiedener Architekturentscheidungen auf die Transkriptionsleistung diskutiert.

Die Vorstellung und Analyse der Ergebnisse für die IAM Handwriting, RIMES und George Washington Datensätze wird seperat in den Abschnitten 5.4.1, 5.4.2 und 5.4.3 vorgenommen. Dafür werden zunächst die Fehlerraten der beiden Architekturen analysiert und, falls vorhanden, mit den Referenzergebnissen ihrer Erfinder verglichen. Für den IAM Handwriting Datensatz folgen darauf weitere Überlegungen bezüglich des Trainingsverlaufs, der Verteilung der Zeichenfehler sowie Versuche zu Veränderungen an der Netzarchitektur von Puigcerver. Abschnitt 5.4.4 widmet sich der Analyse der beiden Netze anhand verschiedener Erweiterungen und Kombinationen der Architekturen. Abschließend enthält Abschnitt 5.4.5 Visualisierungen der Ergebnisse mithilfe von Class Activation Maps. Zur Vereinfachung werden die beiden Architekturen, korrespondierend mit der Darstellung in Kapitel 4, jeweils mit dem ersten Autor des wissenschaftlichen Artikels, Puigcerver bzw. Wigington, bezeichnet.

5.4.1 *Experimente auf dem IAM Datensatz*

Die Ergebnisse für die zeilenbasierte Transkription auf dem Validierungs-Set der IAM Handwriting Database sind in Tabelle 5.4.1 zu sehen. Hinter der jeweiligen Metrik ist das dazugehörige 95%-Konfidenzintervall angegeben und die besten Ergebnisse je Architektur sind **fett** markiert. Für alle Experimente wurden Signifikanztests mit der Nullhypothese $H_0 : CER_1 = CER_2$ bzw. $H_0 : WER_1 = WER_2$ und einem Signifikanzniveau von 5% zum besten Experiment der jeweiligen Architektur durchgeführt. Konnte eine signifikante Abweichung festgestellt werden, wurde das Ergebnis *kursiv* markiert.

Das beste Ergebnis mit einer Zeichenfehlerrate von 4.88% und einer Wortfehlerrate von 17.78% wurde mit der Architektur von Puigcerver in Verbindung mit der RGWD Augmentierung erzielt. Auch für die Architektur von Wigington et al. sind die geringsten Fehlerraten mit der RGWD Augmentierung erreicht worden. In der Zeichenfehlerrate unterscheiden sich die Ergebnisse bei beiden Architekturen zwischen der affinen und RGWD Augmentierung jedoch nur um wenige Zehntel Prozent. Die Signifikanz dieser Leistungsunterschiede konnte daher nicht nachgewiesen werden.

Die Differenzen der Zeichenfehlerraten zwischen den Experimenten mit (RGWD oder affiner) Augmentierung und ohne Augmentierung sind für beide Architekturen signifikant. Dies gilt auch für die Wortfehlerraten der Architektur von Wigington et al. Für die Puigcerver Architektur sind die Unterschiede in der Wortfehlerrate

IAM zeilenbasiert					
Architektur	Augmentierung	CER %		WER %	
Puigcerver	Keine	*5.92*	[5.44 – 6.41]	*20.55*	[19.21 – 21.93]
	Affine	5.21	[4.81 – 5.63]	19.33	[18.09 – 20.65]
	RGWD	**4.88**	[4.48 – 5.28]	**17.78**	[16.57 – 19.05]
Wigington	Keine	*7.40*	[6.85 – 7.96]	*25.98*	[24.49 – 27.57]
	Affine	6.45	[5.96 – 6.94]	23.55	[22.12 – 25.00]
	RGWD	**6.39**	[5.91 – 6.88]	**23.51**	[22.07 – 25.01]
Referenz Puigcerver [Pui17]		4.1	[3.6 – 4.5]	14.6	[13.1 – 16.1]

Tabelle 5.4.1: Zeichen- und Wortfehlerraten in Prozent für die zeilenbasierte Transkription auf dem Validierungs-Set des IAM Handwriting Datensatzes. Hinter den jeweiligen Ergebnissen sind die 95%-Konfidenzintervalle vermerkt.

hingegen nur zwischen den Experimenten mit der RGWD Augmentierung und ohne Augmentierung signifikant. Die Leistungsfähigkeit der Architektur von Puigcerver ist in allen Fällen um 1.24% bis 1.51% CER und 4.22% bis 5.73% WER besser als die der Architektur von Wigington et al. Dies entspricht für beide Metriken eine relative Leistungsdifferenz von ca. 20% bis 30%, deren Signifikanz zwischen den Experimenten mit gleichen Augmentierungsverfahren auch durch die statistischen Tests bestätigt werden konnte.

Das Referenzergebnis aus dem wissenschaftlichen Artikel Puigcervers [Pui17] von 4.1% CER und 14.6% WER ist um ca. 1.1% bzw. 4.7% besser als das Experiment mit der gleichen (affinen) Augmentierung. Eine Erklärung dafür ist, dass Puigcerver zwar auch das vollständige Alphabet des IAM Datensatzes von $|L| = 79$ Zeichen verwendet hat, aber in den hier durchgeführten Experimenten im Gegensatz zum Vorgehen von Puigcerver keine weitere Vorverarbeitung auf den Datensätzen durchgeführt worden ist. Dies wird zwar nicht im wissenschaftlichen Artikel beschrieben, der Programmcode von Puigcerver im verlinkten Github Repository enthält aber eine Vorverarbeitung aller Eingabedaten mit dem Programm imgtxtenh[8] [VRS15], sowie eine Neigungskorrektur mit dem convert Werkzeug von ImageMagick[9]. Auch die Einteilung der Daten in Trainings-, Validierungs- und Test-Sets ist den Angaben zur Anzahl der Beispiele von Puigcerver nach eine andere. Im Repository wird auf eine alternative Einteilung zur offiziellen Partition des Datensatzes verwiesen, die als *Aachens Partition* bezeichnet wird. Aus diesen Gründen sind die Ergebnisse nicht direkt miteinander vergleichbar. Die Evaluation von Wigington et al. enthält für die IAM und RIMES Datensätze lediglich Ergebnisse für die wortbasierte Transkription.

TRAININGSVERLAUF Der zeitliche Verlauf der Trainings-Metriken für die Architektur von Puigcerver mit der RGWD Augmentierung ist in Abbildung 5.4.1 dargestellt. Die Abbildung zeigt die Entwicklung des durchschnittlichen Werts der Kostenfunktion über die Epochen und den dazugehörigen Zeichenfehlerraten auf dem Validierungs-Set. Die durchschnittlichen Kosten der Beispiele einer Epoche starten bei ca. 3.3 und sinken monoton zunächst sehr schnell auf 0.5 bei Epoche 12. Anschließend flacht die Kurve deutlich ab und die Kostenfunktion sinkt erheblich langsamer bis auf 0.034 bei Epoche 166. Die Zeichenfehlerrate auf dem Validierungs-Set beginnt bei 100% nach Epoche 1 und sinkt dann bis Epoche 12 schnell auf einen Wert von ca. 11%. Ähnlich wie der Kurvenverlauf der durchschnittlichen Kosten flacht die Kurve zur Zeichenfehlerrate ab Epoche 12 deutlich ab. Es gibt Schwankungen von etwa $\pm 0.1\%$ die beim

[8] https://github.com/mauvilsa/imgtxtenh
[9] https://imagemagick.org/index.php

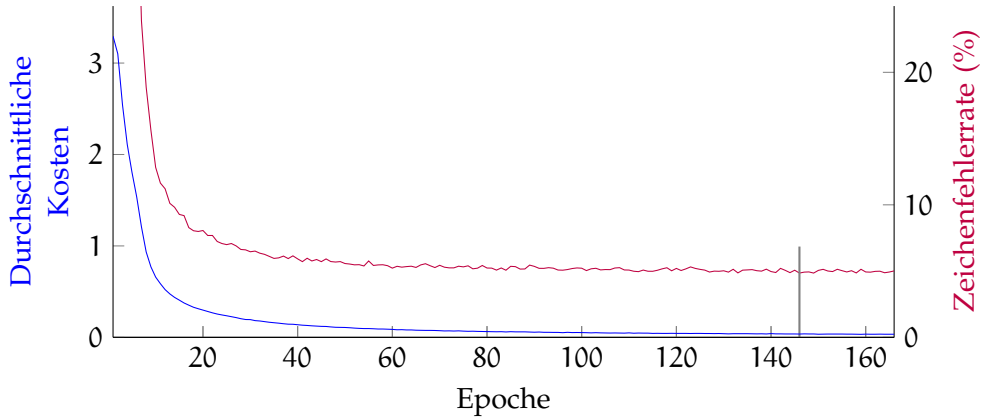


Abbildung 5.4.1: Verlauf des Durchschnitts der CTC Kosten und der Zeichenfehlerrate auf dem Validierungs-Set in jeder Trainingsepoche. Der senkrechte Strich bei Epoche 146 markiert die beste erzielte Zeichenfehlerrate von 4.88%.

Verlauf der Kostenfunktion nicht zu erkennen sind. Die beste Zeichenfehlerrate für dieses Experiment wurde in Epoche 146 erreicht, sodass nach weiteren 20 Epochen der Trainingsvorgang abgebrochen wurde. Ohne und mit der affinen Augmentierung war der Trainingsprozess bereits nach 87 bzw. 96 Epochen abgeschlossen. Die Verlängerung des Trainingsprozesses mit der affinen Augmentierung auf 150 Epochen hatte aber keine weitere Verbesserung der Zeichenfehlerrate auf dem Validierungs-Set zur Folge.

ZEICHENFEHLER Die Zeichen- und Wortfehlerraten geben ein grobes Bild über die Transkriptionsleistung der Verfahren bezogen auf die Gesamtheit der Buchstaben, treffen aber keine Aussage über Zeichen-spezifische Fehler. Eine Zeichenfehlerrate von 10% gibt beispielsweise keine Information darüber, ob gleichmäßig 10% aller Vorkommen der Buchstaben fehlerhaft transkribiert werden, oder ob das Verfahren alle Zeichen bis auf eines (mit einer relativen Häufigkeit von 10%) korrekt erkennt. Neben der Betrachtung der Metriken ist daher auch interessant, welche Zeichen-spezifischen Fehler bei der Transkription besonders häufig auftreten. Hierfür wird die minimale relative Anzahl und eine mögliche Variante von irrtümlichen Ersetzungen, Einfügungen und Entfernungen von Buchstaben gegenüber der Annotation betrachtet, die dem Netz unterlaufen sind. Dabei wird im folgenden für jede Transkription nur einer von potenziell mehreren möglichen Wegen betrachtet, die im Sinne der Levenshtein Distanz optimal sein können. In der vorliegenden Implementierung zur Berechnung der Zeichen- und Wortfehlerraten wird bei mehreren möglichen Operationen die Ersetzung vor der Einfügung und die Einfügung vor der Entfernung eines

Annotation \ Transkription	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	
a	-	0	5	4	20	0	1	0	3	0	0	1	3	8	**56**	1	0	1	2	1	11	0	0	1	0	0	10
b	0	-	0	1	0	1	0	1	0	0	0	3	0	0	1	0	0	0	0	5	0	0	0	0	0	0	1
c	4	0	-	6	4	0	0	0	0	0	0	0	1	1	2	0	0	2	0	0	1	0	0	0	0	1	9
d	0	5	1	-	2	2	1	0	0	0	0	14	0	0	0	0	0	1	6	5	0	0	0	0	0	0	3
e	19	0	6	1	-	0	2	3	1	0	0	2	0	3	11	0	0	2	6	1	2	0	0	0	0	0	26
f	0	0	0	0	0	-	5	0	1	0	0	3	0	0	0	4	0	2	0	9	0	0	0	0	1	0	3
g	1	1	0	0	0	1	-	0	0	1	0	0	0	0	0	7	1	0	3	0	0	0	0	0	4	0	1
h	1	9	0	0	1	5	0	-	3	0	7	17	0	11	0	0	0	4	3	7	1	0	0	0	0	0	11
i	4	0	0	0	2	0	0	0	-	1	0	0	0	1	5	0	0	7	5	1	4	0	0	0	0	0	16
j	0	0	0	0	0	0	0	0	0	-	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
k	0	1	1	1	5	0	0	6	0	0	-	4	0	0	0	0	0	1	0	4	0	0	0	0	0	0	0
l	1	4	0	2	7	3	0	2	0	0	0	-	0	0	1	0	0	1	0	9	0	0	0	0	1	0	8
m	0	0	1	0	1	0	0	0	1	0	1	1	-	11	0	0	0	1	1	0	1	0	4	0	0	0	1
n	3	0	0	0	4	0	1	3	0	0	0	0	20	-	2	0	0	13	2	0	18	0	1	0	0	0	13
o	39	0	1	1	4	0	0	0	0	0	0	0	2	1	-	0	0	0	5	1	7	0	0	0	0	0	5
p	0	1	0	0	2	2	1	0	0	0	0	0	0	0	0	-	0	0	0	1	0	0	0	0	3	0	2
q	1	0	0	0	0	0	2	0	0	0	0	0	0	0	0	1	-	0	0	0	0	0	0	0	0	0	0
r	3	0	9	0	9	1	0	1	5	0	1	3	6	22	2	0	0	-	22	16	0	9	2	1	1	0	**38**
s	7	3	2	0	4	0	1	0	3	1	0	0	3	7	7	3	0	16	-	6	0	2	0	0	1	0	28
t	1	2	4	5	0	9	0	7	0	0	6	20	0	2	0	0	0	0	1	-	0	0	1	0	0	0	4
u	11	0	0	0	9	0	0	0	3	0	0	0	2	20	2	0	0	2	0	1	-	5	1	0	0	0	5
v	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	12	0	0	1	-	0	0	0	0	1
w	0	0	1	1	1	0	0	1	1	0	0	0	0	1	3	0	0	2	1	0	3	2	-	0	0	0	0
x	0	0	1	0	4	0	0	0	0	0	1	0	0	0	0	0	0	4	0	3	0	0	0	-	0	0	1
y	0	0	0	0	0	1	4	0	0	0	0	0	0	0	0	2	0	1	2	0	1	0	0	0	-	0	3
z	0	0	0	0	2	0	1	0	0	0	0	1	0	0	0	0	0	1	1	2	0	0	0	0	0	-	0
	12	1	13	3	13	3	1	2	4	0	0	9	3	12	9	2	0	**14**	8	8	2	0	0	1	1	0	

Tabelle 5.4.2: Anzahl der Vertauschungen von Kleinbuchstaben bei der zeilenbasierten Transkription des Netzes gegenüber der Annotation auf dem IAM Datensatz. Die letzte Spalte bzw. Zeile gibt die Anzahl von Einfügungen bzw. Entfernungen des jeweiligen Buchstabens an.

Zeichens durchgeführt. Diese Fälle treten aber so selten auf, dass das Verhältnis der Operationshäufigkeiten davon weitgehend unabhängig ist. In allen getesteten Fällen entsprechen ca. 75% der 1 641 Fehlerfälle für die Berechnung der Zeichenfehlerrate des Netzes Vertauschungen von Buchstaben. Weitere 15% entfallen auf das Entfernen (Auslassen) von Buchstaben und Einfügen von überschüssigen Buchstaben ist mit 10% der seltenste Fehlerfall.

Die Tabelle 5.4.2 enthält eine Übersicht über die absolute Anzahl der Fehlerfälle für die Kleinbuchstaben. Auffällig ist, dass die Vertauschungen zwischen den Buchstaben *a* und *o* sowie der umgekehrte Fall zwischen *o* und *a* mit 56 und 39 Vorkommen am häufigsten auftreten. Weitere häufige Ersetzungen finden zwischen den Buchstaben *r* und *s* sowie *r* und *n* mit jeweils 22 Vorkommen statt. Die häufigsten Fehler bei den Einfügungen und Entfernungen betreffen das Leerzeichen (34 respektive 44 Mal; nicht in der Tabelle enthalten), gefolgt vom Buchstaben *r* mit 14 irrtümlichen Einfügungen

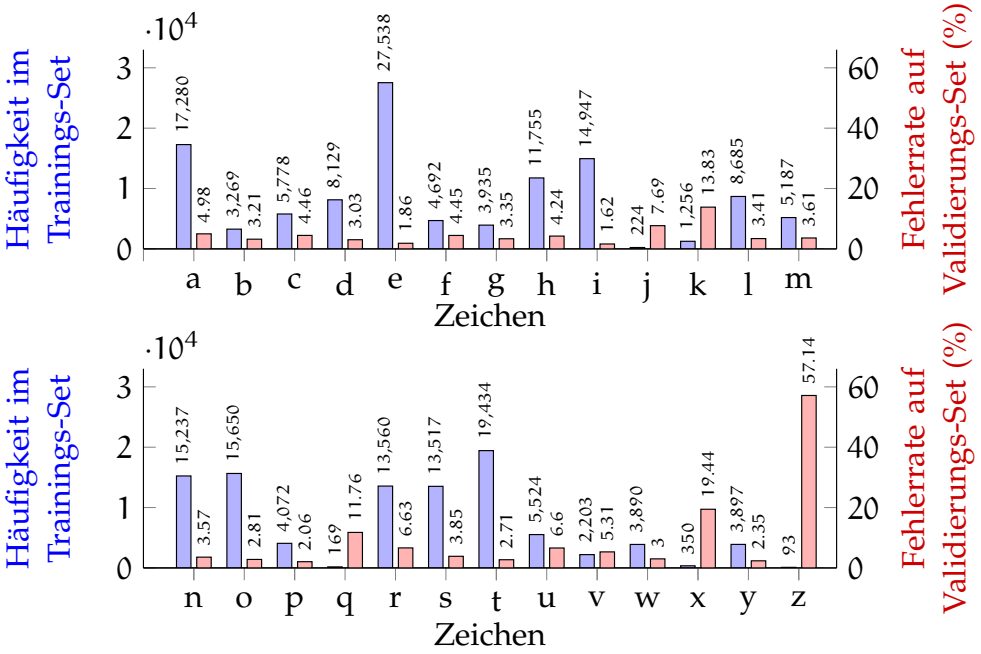


Abbildung 5.4.2: Gegenüberstellung der Häufigkeit des Vorkommens der Kleinbuchstaben im Trainings-Set zu ihrer Fehlerrate auf dem Validierungs-Set des IAM Handwriting Datensatzes. Die Fehlerrate berechnet sich aus der Summe der Ersetzungs-, Einfügungs- und Entfernungsfehler für den jeweiligen Buchstaben bezogen auf die Häufigkeit seines Vorkommens im Validierungs-Set.

und 38 Entfernungen. Anhand der Tabelle können einige Buchstaben identifiziert werden, die in der Handschrift eine große Ähnlichkeit zueinander haben und die deshalb durch das Transkriptionsverfahren häufig vertauscht werden. Neben den bereits genannten zählen dazu beispielsweise die Buchstaben *e* und *a*, *m* und *n* sowie *t* und *l*.

Die große Zahl einiger Buchstabenvertauschungen kann auch auf die Häufigkeit dieser Buchstaben im Trainingsdatensatz zurückgeführt werden. Um den Einfluss dieses Effektes abschätzen zu können, werden in Abbildung 5.4.2 die Vorkommen der Buchstaben im Trainings-Set den Fehlerraten auf dem Validierungs-Set gegenübergestellt. Hier ist auch die negative Korrelation zwischen der Anzahl von Trainingsbeispielen für einen Buchstaben und seiner Fehlerrate sichtbar. Während der Fehleranteil für häufige Buchstaben im Trainings-Set wie *e* und *t* mit 1.86% und 2.71% sehr gering ist, liegt der Fehleranteil für die selten vorkommenden Zeichen wie *k*, *q*, *x* und *z* von 11.76% bis 57.14% erheblich höher. Dies verdeutlicht den großen Bedarf an weiteren Trainingsdaten, da die Kleinbuchstaben im Vergleich zu Großbuchstaben und Sonderzeichen die häufigsten Zeichen in den Datensätzen sind und bestimmte Kleinbuchstaben trotzdem aufgrund ihrer Seltenheit erheblich häufiger falsch transkribiert werden. Für die Transkription dieser Zeichen geringer relativer Häufigkeit ist eine größere Zahl von Beispielen im Trainingsdatensatz daher unerlässlich.

Durch die Unterscheidung der Groß- und Kleinschreibung in den vorliegenden Experimenten besteht die Gefahr der Vertauschung von Groß- und Kleinbuchstaben, insbesondere für Zeichen, die eine visuelle Ähnlichkeit zwischen den beiden Formen

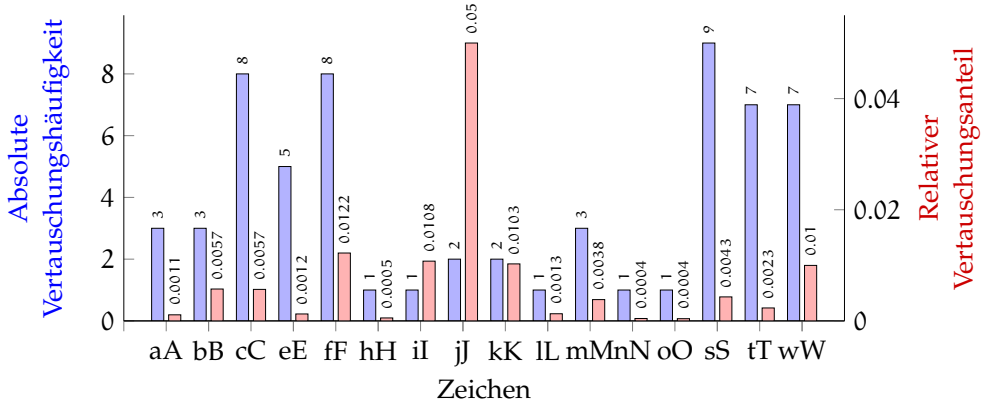


Abbildung 5.4.3: Absolute und relative Anzahl der Vertauschungen zwischen Groß- und Kleinbuchstaben. Bei nicht gezeigten Buchstaben hat keine Verwechslung zwischen den beiden Formen stattgefunden.

aufweisen. Um den Einfluss dieser Fehlerform beurteilen zu können, enthält Abbildung 5.4.3 ein Diagramm über die absolute und relative Häufigkeit der Vertauschung von Groß- und Kleinschreibung zur Anzahl der Vorkommen des jeweiligen Zeichens. Diese Vertauschungen machen nur einen Anteil von ca. 3.8% der 1 641 Zeichenfehler aus. Trotzdem ist erkennbar, dass Zeichen wie *C*, *F* und *S*, deren Groß- und Kleinschreibung einander ähnlich sind, häufiger durch das Transkriptionsverfahren verwechselt werden. Die größte relative Fehlerhäufigkeit betrifft das *J*, bei dem zwei der insgesamt 40 Vorkommen verwechselt worden sind.

WORTBASIERTE TRANSKRIPTION Die Ergebnisse für die wortbasierte Transkription werden in Tabelle 5.4.3 aufgelistet. Abweichend von der Darstellung in Kapitel 5.3 wurde die Batchgröße der Wigington Architektur für die wortbasierte Transkription auf 4 herabgesetzt, da das Training ohne diese Maßnahme nicht erfolgreich abgeschlossen werden konnte. Außerdem wurde Gradient Clamping mit einem Wert von 100 durchgeführt, um numerische Probleme bei der Berechnung der Gradienten in den LSTM-Schichten zu umgehen. Wie in Tabelle 5.4.1 zur zeilenbasierten Transkription werden die jeweils besten Ergebnisse **fett** und die dazu signifikant abweichenden Ergebnisse *kursiv* markiert.

		IAM wortbasiert			
Architektur	Augmentierung	CER %		WER %	
Puigcerver	Keine	*8.41*	[7.99 – 8.86]	*22.27*	[21.26 – 23.30]
	Affine	**6.85**	[6.45 – 7.25]	**18.87**	[17.92 – 19.83]
	RGWD	7.07	[6.66 – 7.46]	19.27	[18.31 – 20.23]
Wigington*	Keine	*9.79*	[9.33 – 10.24]	*26.28*	[25.21 – 27.40]
	Affine	8.76	[8.33 – 9.19]	24.02	[22.96 – 25.04]
	RGWD	**8.71**	[8.27 – 9.14]	**23.80**	[22.78 – 24.87]
Referenz Wigington et al. [WSD+17]		6.07		19.07	

*Für die wortbasierte Transkription musste abweichend die Batchsize auf 4 reduziert und Gradient Clamping eingesetzt werden, damit der Trainingsprozess erfolgreich abgeschlossen werden konnte.

Tabelle 5.4.3: Zeichen- und Wortfehlerraten in Prozent für die zeilenbasierte Transkription auf dem IAM Handwriting Datensatz. Hinter den jeweiligen Ergebnissen sind die 95%-Konfidenzintervalle vermerkt.

Insgesamt sind die Zeichenfehlerraten bei der wortbasierten Transkription um 1.64% bis 2.49% höher als bei der zeilenbasierten Transkription. Auch die meisten Wortfehlerraten liegen um 1.49% bis 1.72% (Puigcerver) bzw. 0.29% bis 0.47% (Wigington) über denen der zeilenbasierten Verfahren. Lediglich die Wortfehlerrate des Experiments zur zeilenbasierten Transkription mit der Architektur von Puigcerver und affiner Augmentierung ist um 0.46% höher als die Wortfehlerrate des gleichen Experiments zur wortbasierten Transkription. Wie bei der zeilenbasierten Transkription konnten mit der Puigcerver Architektur um durchschnittlich 1.64% bessere Zeichenfehlerraten und um 4.56% bessere Wortfehlerraten gegenüber der Wigington Architektur erzielt werden. Auch die statistischen Tests bestätigen die höhere Transkriptionsleistung der Architektur von Puigcerver im Vergleich zu den Experimenten mit der Wigington Architektur und der gleichen Augmentierungsfunktion.

Im Gegensatz zur zeilenbasierten Transkription zeigte sich die affine Augmentierung bei der Puigcerver Architektur als die leistungsfähigste Variante mit Zeichen- und Wortfehlerraten von 6.85% und 18.87%. Bei der Wigington Architektur wurde hingegen mit der RGWD Augmentierung die beste Zeichen- und Wortfehlerrate von 8.71% bzw. 23.80% erzielt. Die Unterschiede zwischen affiner und RGWD Augmentierung sind wie bei der zeilenbasierten Transkription für beide Architekturen nur gering, sodass die Signifikanz der Leistungsdifferenz zwischen diesen Experimenten nicht nachgewiesen werden konnte. Zwischen den Verfahren ohne Augmentierung und mit (RGWD oder affiner) Augmentierung existiert jedoch ein statistisch signifikanter Leistungsunterschied für die Zeichen- und Wortfehlerraten beider Architekturen.

Der Leistungsvorteil zeilenbasierter Transkription und die daraus resultierende geringere Zeichenfehlerrate kann durch den Gewinn von Kontext begründet werden, der mit der Eingabe ganzer Zeilen einhergeht. Das rekurrente Neuronale Netz kann innerhalb der zeilenbasierten Transkription auch die Wortumgebung betrachten und hat so z. B. einen Vergleich für die Transkription mehrdeutiger Zeichen. Insbesondere tiefe rekurrente Netze benötigen eine gewisse Anzahl von Sequenzelementen in der Eingabe, um von ihren rekurrenten Verbindungen profitieren zu können.

Da die Anzahl der Zeichenfehler innerhalb eines Wortes für die Berechnung der Wortfehlerrate keinen Einfluss hat, haben Fehler bei der Erkennung von Wortgrenzen einen größeren Effekt und führen bei den zeilenbasierten Transkriptionsverfahren zu einer höheren Wortfehlerrate. Dahingegen können bei der wortbasierten Transkription bezüglich der Wortgrenzen keine Fehler auftreten. Dies erklärt die geringere Leistungsdifferenz zwischen den zeilen- und wortbasierten Verfahren bezüglich der Wortfehlerrate gegenüber der Zeichenfehlerrate.

Das Referenzergebnis von 6.07% CER und 19.07% WER sind die einzigen Fehlerraten von lexikonfreier Transkription, die Wigington et al. [WSD+17] in ihrem

wissenschaftlichen Artikel zum IAM Datensatz angeben. Hier wurde jedoch ein Alphabet ohne Sonderzeichen und ohne Beachtung von Groß- und Kleinschreibung verwendet. Außerdem wandten die Autoren neben der RGWD Augmentierung im Training auch Test-Side Augmentation (siehe Abschnitt 2.4.1) und Profilnormalisierung an, was eine Erklärung für die Differenz der Zeichen- und Wortfehlerraten zum hier durchgeführten Experiment ist. Wigington et al. zeigen mit Ergebnissen zur lexikonbasierten Transkription, dass Test-Side Augmentation den größten positiven Effekt auf die Erkennungsleistung hat. Ein weiteres Experiment mit der Wigington Architektur, affiner Augmentierung und einem Alphabet bestehend aus Kleinbuchstaben und Ziffern ergab eine Zeichen- und Wortfehlerraten von 8.04% [7.63 – 8.47] bzw. 24.26% [23.18 – 25.37]. Allein die Verkleinerung des Alphabets führte somit zu einer statistisch signifikanten Verbesserung der Zeichenfehlerrate um ca. 0.7%.

NETZAKTIVIERUNGEN Die Idee für das heuristische Dekodierungsverfahren von Graves et al. (siehe Kapitel 2.6.3) basiert auf starken Blank-Prädiktionen des Netzes, die durch kurzzeitige Peaks für die erkannten Zeichen unterbrochen werden. Dieses Verhalten konnte exemplarisch anhand des in der Abbildung 5.4.4 gezeigten Beispiels bestätigt werden. Es zeigt für jeden Zeitschritt die Aktivierung des Neuronalen Netzes für die Zeichen *n*, *e*, *x* und *t* sowie das Blank-Symbol (–). Alle weiteren Zeichen des Alphabets, deren Aktivierungen nahe Null liegen, wurden in der Grafik ausgelassen. In den meisten Zeitschritten korrespondiert die stärkste Aktivierung mit dem Blank-

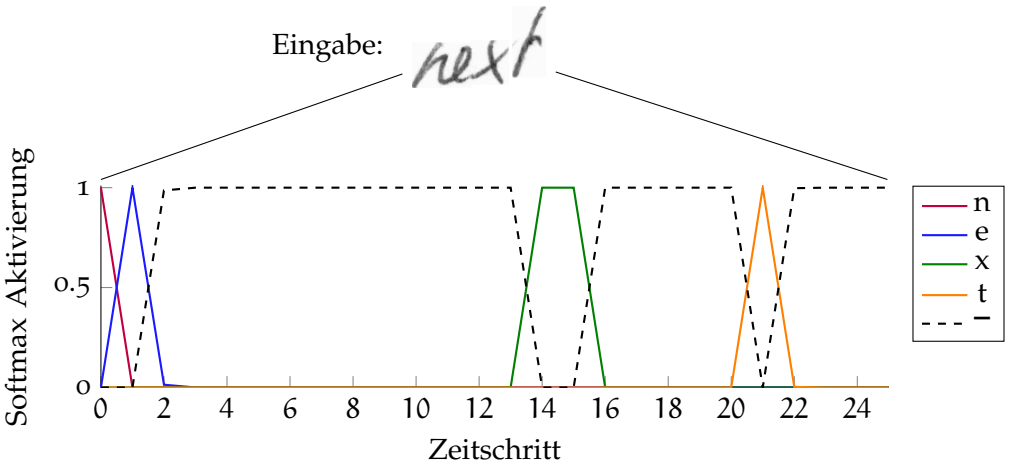


Abbildung 5.4.4: Verlauf der Aktivierungen für die Zeichen *n*, *e*, *x*, *t* und Blank (–) je Zeitschritt für ein Beispielbild aus dem IAM Datensatz [MB02].

Symbol (–), die nur vereinzelt durch starke Aktivierungen zu den anderen Zeichen unterbrochen wird. Diese Unterbrechungen umfassen meist nur einen Zeitschritt, mit Ausnahme des Zeichens *x*, das im vorliegenden Fall zwei Zeitschritte einnimmt. Im Beispiel sind somit die Voraussetzungen für das effiziente Dekodierungsverfahren von Puigcerver gegeben.

ZEICHENFEHLER IN ABHÄNGIGKEIT ZUR WORTLÄNGE Die IAM Database enthält Wörter mit 1 bis 16 Buchstaben, wobei Wörter der Länge 3 am häufigsten sind. Die Abbildung 5.4.5 zeigt die Häufigkeit von Wörtern und die durchschnittliche Zeichenfehlerrate je Wortlänge. Es ist zu erkennen, dass die Zeichenfehlerrate für Wörter der Länge 2 am geringsten ist. Dies wird wahrscheinlich dadurch verursacht, dass im Validierungs- und Trainings-Set 45 bzw. 126 Wörter der Länge zwei enthalten sind und das Netz somit die Möglichkeit hatte, ein implizites Sprachmodell zu lernen. Es existieren auch nur 110 bzw. 338 verschiedene Wörter der Länge 3 im Validierungs- und Trainings-Set. Für Wörter der Länge 5 bis 10 ist die Zeichenfehlerrate relativ konstant bei ca. 8%, bevor sie für die langen Wörter mit 11 bis 16 Zeichen stark zu schwanken beginnt. Die Aussagekraft der Zeichenfehlerraten dieses Diagrammbereichs muss jedoch infrage gestellt werden, da die Anzahl von Beispielen der Länge $\geqslant 11$ ausgehend von 80 Beispielen mit jedem zusätzliche Buchstaben etwa halbiert wird und damit insgesamt sehr gering ist.

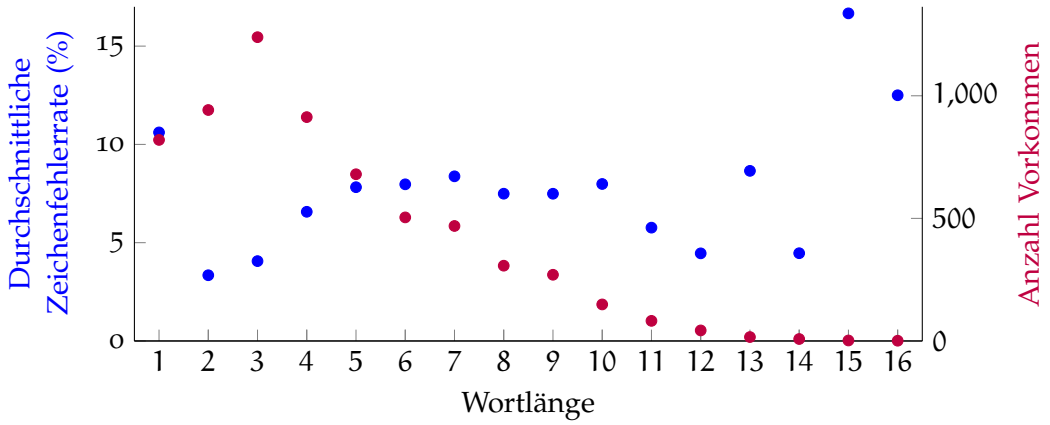


Abbildung 5.4.5: Darstellung der durchschnittlichen Zeichenfehlerrate und Anzahl der Vorkommen für alle vorkommenden Wortlängen in der IAM Database [MB02].

ALTERNATIVE NETZARCHITEKTUREN UND AUGMENTIERUNGSOPERATIONEN

Weitere Experimente mit verschiedenen Veränderungen an der Architektur und den Augmentierungsoperationen werden in Tabelle 5.4.4 zusammengefasst. Alle Tests wurden, sofern nicht anders angegeben, mit der Architektur von Puigcerver und der RGWD Augmentierung auf dem IAM Datensatz für die zeilenbasierte Transkription durchgeführt. Die erste Zeile enthält das Vergleichsergebnis für die RGWD Augmentierung aus Tabelle 5.4.1. Für die Netzarchitektur des Experiments in Zeile zwei wurde die LeakyReLu Aktivierungsfunktion der Faltungsblöcke durch die Exponential Linear Units ausgetauscht (siehe Kapitel 2.4.2). Es kann kein positiver Effekt durch die Verwendung von ELU Aktivierungsfunktion festgestellt werden. Die Zeichen- und Wortfehlerraten der Referenz sind um ca. 0.7% bzw. 2% geringer. Dieser Unterschied ist jedoch nicht statistisch signifikant.

Der Austausch der Long Short-Term Memory Schichten durch Gated Recurrent Units in den rekurrenten Blöcken des CRNNs reduziert die Parameteranzahl insgesamt von 9.6 auf 7.2 Millionen. Hierdurch verschlechtert sich die Zeichen- und Wortfehlerrate der Architektur auf dem IAM Datensatz zur zeilenbasierten Transkription jedoch signifikant um 1.5% bzw. 5.3%.

Die Verwendung von bidirektionalen rekurrenten Schichten ermöglicht es dem rekurrenten Teil der CRNN-Architektur sowohl vorherige als auch folgende Merkmale der Sequenz zu betrachten und für die Transkription des aktuellen Sequenzelements miteinzubeziehen. Die Limitierung auf vorhergehenden Kontext mit einer unidirektionalen LSTM Architektur führt zu einer erheblichen Reduktion der Parameteranzahl

Architektur- und Augmentierungsexperimente

Experiment	CER %		WER %	
Referenz (RGWD)	4.88	[4.48 – 5.28]	17.78	[16.57 – 19.05]
ELU Aktivierungsfunktion	5.24	[4.81 – 5.67]	18.57	[17.31 – 19.86]
Gated Recurrent Units	*6.41*	[5.92 – 6.92]	*22.99*	[21.55 – 24.45]
Unidirektionales LSTM	*7.59*	[7.06 – 8.15]	*27.11*	[25.54 – 28.72]
RGWD + Affine Augmentierung	**4.73**	[4.36 – 5.10]	**17.78**	[16.55 – 19.00]
Reduziertes Alphabet ($\|L\| = 37$)	**4.68**	[4.28 – 5.09]	**16.66**	[15.44 – 17.89]

Tabelle 5.4.4: Zeichen- und Wortfehlerraten in Prozent für Experimente mit verschiedenen Netzarchitekturen und Augmentierungsverfahren für die zeilenbasierte Transkription auf dem IAM Datensatz. Bessere Ergebnis als das Referenzergebnis sind **fett** und davon signifikant abweichende Ergebnisse sind *kursiv* markiert.

auf 3.8 Millionen, aber auch einer deutlichen Verschlechterung der Metriken um 2.71% CER und 9.33% WER. Damit ist die Architektur mit unidirektionalen LSTMs signifikant schlechter als das Referenz- und GRU-Ergebnis.

Die bisherigen Experimente haben gezeigt, dass die Verwendung von Augmentierungsoperationen im Training die Erkennungsleistung verbessern kann. Je nachdem, ob wort- oder zeilenbasierte Transkription durchgeführt und welche Netzarchitektur verwendet wurde, führte die RGWD oder die affine Augmentierung zu den besten Zeichen- und Wortfehlerraten. Die Verbindung beider Verfahren, sodass das Wort- oder Zeilenabbild zunächst mit dem RGWD Verfahren verzerrt und anschließend affin augmentiert wird, verbesserte die Zeichenfehlerrate der zeilenbasierten Transkription auf dem IAM Datensatz nochmals um 0.15%. Die Wortfehlerrate ist gegenüber dem Referenzergebnis jedoch identisch geblieben und die Differenzen sind sowohl für die Wort- als auch für die Zeichenfehlerrate nicht statistisch signifikant.

Alle vorhergehenden Experimente verwendeten den vollständigen Zeichensatz der IAM Handwriting Database von 79 Zeichen. Je nach Wettbewerb oder Präferenz der Forscher wird für die Evaluation teils ein kleineres Alphabet beispielsweise ohne Sonderzeichen oder Unterscheidung von Groß- und Kleinschreibung verwendet. Um den Einfluss der Alphabetgröße zu testen, wurde ein Experiment mit reduziertem Alphabet durchgeführt, das lediglich die Kleinbuchstaben *a-z*, Ziffern *0-9* und das Leerzeichen enthält. Mit diesem Alphabet verbesserten sich die Zeichen- und Wortfehlerraten gegenüber der Referenz um 0.2% bzw. 1.12%, da das Verfahren nur eine kleinere Anzahl von Zeichen unterscheiden muss. Diese Verbesserung ist allerdings nicht statistisch signifikant.

5.4.2 *Experimente auf dem RIMES Datensatz*

Die Ergebnisse zur zeilenbasierten Transkription auf dem RIMES Datensatz sind in Tabelle 5.4.5 angegeben. Wie zuvor entsprechen **fett** markierte Werte den besten Fehlerraten für die jeweilige Architektur und *kursiv* markierte Werte signifikant vom besten Fall abweichende Ergebnisse.

In diesem Experiment zeigte die Architektur von Puigcerver für alle Augmentierungsvarianten eine um durchschnittlich 0.9% geringere Zeichen- und um 4.56% geringere Wortfehlerrate. Die Differenzen der Wortfehlerraten für die beiden Architekturen sind statistisch signifikant, die Signifikanz der Unterschiede in den Zeichenfehlerraten konnte jedoch nicht bestätigt werden. Im Vergleich zur zeilenbasierten Transkription auf dem IAM Datensatz sind die Zeichen- und Wortfehlerraten in den hier durch-

RIMES zeilenbasiert					
Architektur	Augmentierung	CER %		WER %	
Puigcerver	Keine	4.52	[3.71 – 5.42]	*16.22*	[14.69 – 17.84]
	Affine	3.91	[3.12 – 4.83]	14.53	[13.04 – 16.11]
	RGWD	**3.81**	[3.02 – 4.74]	**13.14**	[11.71 – 14.65]
Wigington	Keine	5.72	[4.87 – 6.68]	*22.21*	[20.50 – 23.93]
	Affine	4.70	[3.90 – 5.63]	18.18	[16.62 – 19.85]
	RGWD	**4.53**	[3.72 – 5.44]	**17.19**	[15.63 – 18.83]
Referenz Puigcerver [Pui17]		3.0	[2.6 – 3.5]	12.4	[10.9 – 14.0]

Tabelle 5.4.5: Zeichen- und Wortfehlerraten in Prozent für die zeilenbasierte Transkription auf dem RIMES Datensatz. Hinter den jeweiligen Ergebnissen sind die 95%-Konfidenzintervalle vermerkt.

geführten Experimenten um durchschnittlich 1.5% bzw. 5.2% geringer, was auf eine geringere Komplexität des Datensatzes schließen lässt.

Auch für den RIMES Datensatz verwendet Puigcerver das vollständige Alphabet mit $|L| = 99$ Zeichen, führt aber eine Vorverarbeitung der Eingabedaten mit imgtxtenh und ImageMagick durch. Dies erklärt die Differenz von 0.91% CER und 2.13% WER im Vergleich zum hier durchgeführten Experiment mit der affinen Augmentierung.

WORTBASIERTE TRANSKRIPTION Tabelle 5.4.6 enthält die Ergebnisse zur wortbasierten Transkription auf dem RIMES Datensatz. Im Gegensatz zu den anderen wortbasierten Experimenten musste die Batchsize für die Trainingsvorgänge auf dem RIMES Datensatz nicht auf 4 reduziert und kein Gradient Clamping durchgeführt werden. In diesem Experiment konnte mit der Architektur von Wigington et al. die beste Zeichen- und Wortfehlerrate von 4.4% respektive 15.05% erreicht werden. Mit Ausnahme des Versuchs ohne Augmentierung liegen beide Fehlerraten durchschnittlich ca. 0.5% unterhalb denen der Architektur von Puigcerver. Für die Zeichenfehlerraten in den Experimenten mit Affiner und RGWD Augmentierung sind die Leistungsdifferenzen zwischen den beiden Architekturen auch statistisch signifikant.

Ähnlich zur wortbasierten Transkription auf dem IAM Datensatz wurden bei beiden Architekturen mit der affinen Augmentierung die besten Ergebnisse erzielt. Die Unterschiede zu den Ergebnissen mit der RGWD Augmentierung sind jedoch gering

RIMES wortbasiert					
Architektur	Augmentierung	CER %		WER %	
Puigcerver	Keine	5.26	[4.93 – 5.59]	16.29	[15.45 – 17.12]
	Affine	**4.98**	[4.67 – 5.30]	**15.58**	[14.76 – 16.41]
	RGWD	4.99	[4.68 – 5.29]	15.90	[15.10 – 16.73]
Wigington	Keine	*5.18*	[4.88 – 5.49]	*17.39*	[16.55 – 18.26]
	Affine	**4.40**	[4.13 – 4.68]	**15.05**	[14.24 – 15.86]
	RGWD	4.50	[4.23 – 4.78]	15.30	[14.50 – 16.12]
Referenz Wigington et al. [WSD+17]		3.09		11.29	

Tabelle 5.4.6: Zeichen- und Wortfehlerraten in Prozent für die wortbasierte Transkription auf dem RIMES Datensatz. Hinter den jeweiligen Ergebnissen sind die 95%-Konfidenzintervalle vermerkt.

und nicht statistisch signifikant. Für die Architektur von Wigington et al. ist das Ergebnis des Experiments ohne Augmentierung signifikant schlechter als die Ergebnisse der beiden Fälle mit Augmentierung. Für die Puigcerver Architektur konnte keine statistische Signifikanz der Leistungsdifferenzen zwischen den drei Fällen festgestellt werden.

Das Referenzergebnis von Wigington et al. wurde wie bei der wortbasierten Transkription auf dem IAM Datensatz mit einem Alphabet ohne Sonderzeichen, ohne Beachtung von Groß- und Kleinschreibung und unter Verwendung weiterer Techniken wie Test-Side Augmentation und Profilnormalisierung erzielt. Dies ist eine Erklärung für die Differenz der Zeichenfehlerraten von ca. 1.5% und Wortfehlerraten von 4% zum hier durchgeführten Experiment. Wigington et al. geben zusätzlich detaillierte Ergebnisse zu allen Kombinationsmöglichkeiten von RGWD Augmentierung, Test-Side Augmentation und Profilnormalisierung zur Bewertung des jeweiligen Einflusses auf die Fehlerraten an. Nur mit der RWGD Augmentierung im Training und ohne Test-Side Augmentation und Profilnormalisierung erreichten Wigington et al. Zeichen- und Wortfehlerraten von 2.43% und 5.12% auf dem RIMES Datensatz. Diese Ergebnisse sind aber lexikonbasiert und daher nicht mit den hier durchgeführten Experimenten vergleichbar.

5.4.3 *Experimente auf dem George Washington Datensatz*

Die Ergebnisse für die wortbasierte Transkription auf dem George Washington Datensatz [FKFB12] sind in Tabelle 5.4.7 dargestellt. In diesem Experiment konnte mit der Architektur von Wigington et al. das beste Ergebnis mit einer Zeichenfehlerrate von 2.89% und einer Wortfehlerrate von 8.01% erzielt werden. Da die Differenzen der Zeichenfehlerraten nur maximal 0.31% und die der Wortfehlerraten maximal 1.95% für die jeweiligen Augmentierungsformen betragen, sind die Unterschiede zwischen den Architekturen in diesem Experiment nicht statistisch signifikant. Innerhalb der Wigington Architektur konnte die Signifikanz der Leistungsdifferenzen in den einzelnen Augmentierungsoperationen nur zwischen dem Experiment ohne und mit affiner Augmentierung nachgewiesen werden. Wie in den meisten Experimenten zur wortbasierten Transkription auf den IAM Handwriting und RIMES Datensätzen lieferte die affine Augmentierung für den George Washington Datensatz die besten Ergebnisse.

Die hier ermittelten Zeichen- und Wortfehlerraten liegen unterhalb der auf den IAM und RIMES erzielten Werte. Dies kann mit der Homogenität der George Washington Database erklärt werden. Sie umfasst die kleinste Menge an Trainingsdaten und enthält nur die Handschrift einer einzelnen Person, sodass die Intraklassenvarianz erheblich geringer ist als die der anderen Datensätze. Die Gestalt der Handschrift aus den Beispielen des Validierungs-Sets der George Washington Database ist den Verfahren

Washington wortbasiert					
Architektur	Augmentierung	CER %		WER %	
Puigcerver	Keine	4.42	[3.22 – 5.79]	12.30	[9.57 – 15.23]
	Affine	**3.20**	[2.15 – 4.39]	**8.79**	[6.45 – 11.33]
	RGWD	3.88	[2.70 – 5.20]	10.16	[7.62 – 12.89]
Wigington*	Keine	*4.65*	[3.47 – 5.96]	*13.87*	[10.94 – 16.80]
	Affine	**2.89**	[1.85 – 4.15]	**8.01**	[5.66 – 10.35]
	RGWD	3.88	[2.83 – 5.09]	12.11	[9.38 – 15.04]

*Für die wortbasierte Transkription musste abweichend die Batchsize auf 4 reduziert und Gradient Clamping eingesetzt werden, damit der Trainingsprozess erfolgreich abgeschlossen werden konnte.

Tabelle 5.4.7: Zeichen- und Wortfehlerraten in Prozent für die wortbasierte Transkription auf dem George Washington Datensatz. Hinter den jeweiligen Ergebnissen sind die 95%-Konfidenzintervalle vermerkt.

bereits aus dem Training bekannt. Die geringe Anzahl an Validierungsbeispielen verursacht die breiteren Konfidenzintervalle.

5.4.4 *Vergleich der Architekturen*

Die Architekturen von Puigcerver und Wigington et al. zeigen in den hier getesteten Fällen teils deutliche Differenzen in der Leistungsfähigkeit. Sie unterscheiden sich sowohl im Aufbau der Faltungs- und rekurrenten Schichten als auch in der Anzahl von Parametern. Daher soll in den folgenden Experimenten der Effekt dieser Architekturentscheidungen analysiert werden. Die ersten beiden Zeilen der Tabelle 5.4.8 enthalten zum Vergleich die jeweils besten Ergebnisse der beiden Neuronalen Netze zur zeilenbasierten Transkription auf dem IAM Datensatz mit der RGWD Augmentierung.

Das Experiment mit der Bezeichnung *Puigcerver Erweitert* wurde mit dem von Puigcerver vorgesehenen Aufbau und Trainings-Setup durchgeführt. Die einzige Veränderung betrifft die Anzahl der LSTM Zellen im rekurrenten Teil des Netzes, die von 256 auf 400 erhöht wurde. Dies vergrößert bei 128 Pixel hohen Eingabebildern die Gesamtanzahl der Parameter von 9.6 auf 20.9 Millionen und damit ungefähr auf das Niveau der Architektur von Wigington et al. Umgekehrt wurde für das Experiment mit der Bezeichnung *Wigington Reduziert* ausgehend vom Referenz-Setup von Wigington et al. die Anzahl der LSTM Zellen für beide rekurrente Schichten auf 200 reduziert. Dies verringert die Gesamtanzahl der Parameter von 19.8 auf 9.9 Millionen, was nah an der Parameteranzahl der Architektur von Puigcerver liegt.

Die nächsten beiden Zeilen der Tabelle enthalten Experimente zu Kombinationen der Faltungs- und rekurrenten Teile der beiden Architekturen. Für das Experiment *P-CNN* $\rightarrow$ *W-RNN* wurden die Faltungsblöcke der Architektur von Puigcerver mit den rekurrenten Blöcken von Wigington et al. zu einem neuen CRNN verbunden. In diesem Fall wurden die Eingabebilder für das beste Ergebnis auf 128 Pixel normalisiert und das Netz mit einer Lernrate von $3 \cdot 10^{-4}$ trainiert. Umgekehrt entspricht das Experiment *W-CNN* $\rightarrow$ *P-RNN* der Kombination aus den Faltungsblöcken der Wigington-Architektur und den rekurrenten Schichten von Puigcerver. Hier konnte das beste Ergebnis durch eine Höhennormalisierung auf 80 Pixel und mit einer niedrigeren Lernrate von $5 \cdot 10^{-5}$ erreicht werden. Die Faltungsblöcke scheinen demnach maßgebend für die Wahl der Trainingsparameter zu sein.

Die letzten beiden Zeilen der Tabelle betreffen Experimente *Puigcerver 6-LSTMs* und *Puigcerver 8-LSTMs* zur Tiefe des rekurrenten Teils der Architektur von Puigcerver, die von 5 auf 6 bzw. 8 LSTM-Schichten erweitert wurde.

Architekturveränderungen					
Experiment	Parameter ($\cdot 10^6$)	CER %		WER %	
Puigcerver Referenz	9.6	**4.88**	[4.48 – 5.28]	**17.78**	[16.57 – 19.05]
Wigington Referenz	19.8	*6.39*	[5.91 – 6.88]	*23.51*	[22.07 – 25.01]
Puigcerver Erweitert	20.9	5.35	[4.92 – 5.79]	*19.62*	[18.36 – 20.91]
Wigington Reduziert	9.9	*6.35*	[5.87 – 6.85]	*23.81*	[22.31 – 25.33]
P-CNN → W-RNN	10.1	*6.41*	[5.90 – 6.94]	*22.99*	[21.52 – 24.52]
W-CNN → P-RNN	16.6	*6.07*	[5.59 – 6.54]	*21.90*	[20.56 – 23.31]
Puigcerver 6-LSTMs	11.2	4.94	[4.55 – 5.35]	17.76	[16.55 – 19.01]
Puigcerver 8-LSTMs	14.3	*5.72*	[5.30 – 6.16]	*20.01*	[18.75 – 21.27]

Tabelle 5.4.8: Zeichen- und Wortfehlerraten für Experimente mit verschiedenen Netzarchitekturen und Augmentierungsverfahren für die zeilenbasierte Transkription auf dem IAM Datensatz. Das beste Ergebnis ist **fett** und davon signifikant abweichende Ergebnisse sind *kursiv* markiert.

Die Erweiterung der Puigcerver Architektur auf über 20 Millionen Parameter führte zu einer Erhöhung der Zeichen- und Wortfehlerrate um 0.47% bzw. 1.84%. Die Signifikanz der Leistungsdifferenz konnte durch einen Hypothesentest zwar nur für die Wortfehlerrate bestätigt werden, die höheren Fehlerraten lassen aber darauf schließen, dass die größere Parameteranzahl nicht zu einem leistungsfähigeren Verfahren führt. Wahrscheinlich erfährt das erweiterte Modell im Training trotz des RGWD Augmentierungsverfahrens eine stärkere Überanpassung. Da größere Netze oft mehr Epochen bis zur Konvergenz des Trainings benötigen, hat die Verwendung des gleichen Early-Stopping Zeitpunkts wahrscheinlich ebenfalls einen negativen Effekt.

Umgekehrt erzielte die auf 9.9 Millionen Parameter reduzierte Architektur von Wigington et al. nahezu die gleichen Zeichen- und Wortfehlerraten wie das nicht reduzierte Referenzverfahren. Auch wenn die Leistungsfähigkeit der Architektur durch die Parameterreduktion nicht verbessert werden konnte, wurde sie durch die verringerte Modellkapazität auch nicht signifikant verschlechtert. Der große Umfang von 20.9 Millionen Parametern hat also keinen positiven Einfluss auf die Erkennungsleistung des Verfahrens im Vergleich zur reduzierten Architektur.

Während die Kombination der Faltungsblöcke von Puigcerver mit den zwei rekurrenten Blöcken von Wigington et al. zu einer Transkriptionsleistung führt, die sich nicht signifikant vom Referenzergebnis von Wigington et al. unterscheidet, konnte mit

der umgekehrten Variante eine leichte Verbesserung der Zeichenfehlerrate um 0.32% und der Wortfehlerrate um 1.61% erreicht werden. Auch diese Differenz ist nicht statistisch signifikant und das Ergebnis liegt immer noch um 1.19% bzw. 1.61% unterhalb der Zeichen- und Wortfehlerrate des Referenzergebnisses von Puigcerver. Trotzdem ist erkennbar, dass die fünf sequentiell angeordneten LSTM-Schichten der Puigcerver Architektur auf dem hier getesteten Datensatz einen Leistungsvorteil gegenüber der Variante mit zwei LSTM-Schichten und (teilweise) mehr LSTM Zellen von Wigington et al. liefern.

Eine weitere Erhöhung der Anzahl von LSTM-Schichten im Experiment *Puigcerver 6-LSTMs* konnte keine signifikante Veränderung der Transkriptionsleistung des Verfahrens herbeiführen. Das Netz mit 8 LSTM-Schichten *Puigcerver 8-LSTMs* zeigte sogar deutlich schlechtere Zeichen- und Wortfehlerraten als die Referenz. Die optimale Tiefe des rekurrenten Teils der aktuellen CRNN Architektur scheint somit 5 zu sein, wie von Puigcerver vorgesehen.

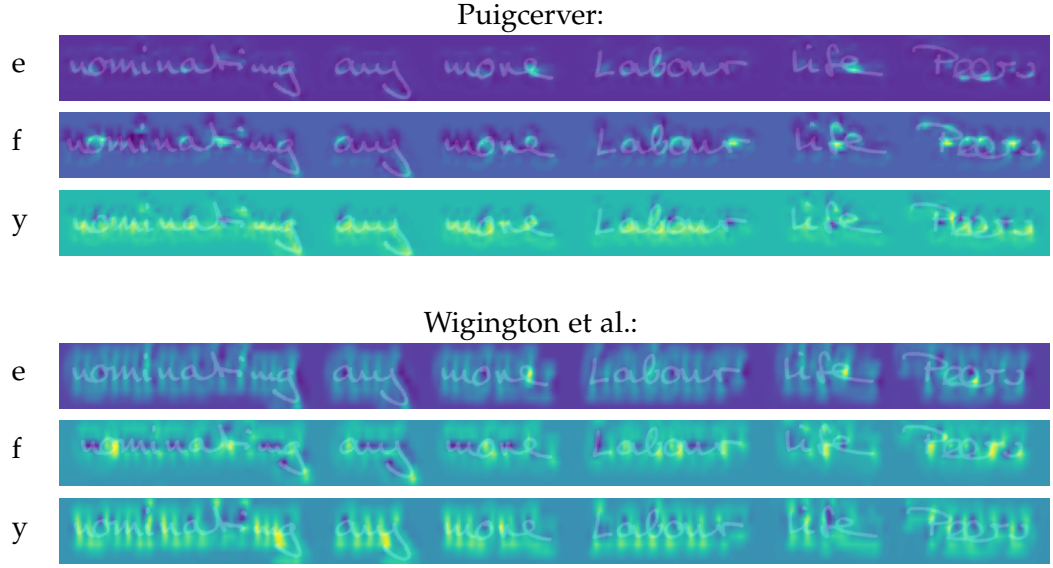
Die Gestaltung der rekurrenten Blöcke beider Netzarchitekturen ist auch eine Erklärung für die teils unterschiedlichen Ergebnisse der zeilen- und wortbasierte Transkription. Während die Architektur von Puigcerver mit ihren fünf LSTM-Schichten bei der zeilenbasierten Transkription auf den IAM und RIMES Datensätzen bessere Ergebnisse erzielen konnte, zeigte die Architektur von Wigington et al. bei der wortbasierten Transkription auf den RIMES und George Washington Datensätzen eine höhere Leistungsfähigkeit. Um von den rekurrenten Verbindungen der tiefen LSTM-Architektur von Puigcerver zu profitieren, ist demnach eine gewisse Anzahl von Sequenzelementen in der Eingabe notwendig. Diese ist bei der zeilenbasierten Transkription natürlich größer als bei der wortbasierten Transkription.

5.4.5 *Class Activation Maps*

Wie in Kapitel 4.4 beschrieben, können Class Activation Maps dazu eingesetzt werden die Aktivierungen der Faltungsschichten für die einzelnen Zeichen des Alphabets auf dem Eingabebild zu betrachten, um einen Einblick in den Informationsfluss zwischen den Faltungs- und rekurrenten Blöcken der CRNN-Architektur zu erhalten. Dazu wurden die beiden Netze von Puigcerver und Wigington et al. zur wortbasierten Transkription auf dem IAM Datensatz um die Class Activation Architektur erweitert. Nach dem Training der voll-verbundenen Class Activation Schicht wurden beispielhaft die Grafiken in Abbildung 5.4.6 erzeugt, indem die Heatmaps mit einer Deckkraft von 90% über die Eingaben gelegt wurden.

Die Grafik zeigt die Class Activation Heatmaps für beide Architekturen sowie eine Auswahl von Buchstaben. Es ist zu erkennen, dass die durch die Heatmaps hervorgehobenen Bildbereiche für den Buchstaben *e* bei beiden Architekturen mit den Positionen des Buchstabens im Eingabebild korrespondieren. Zwar sind die Aktivierungen im Beispielbild für die Architektur von Puigcerver etwas schwächer im Vergleich zur Wigington Architektur, dafür enthält letztere eine schwache Aktivierung in allen Bereichen, in denen Schrift enthalten ist. Trotz der Verwendung von Average Pooling überdecken die hervorgehobenen Bereiche in beiden Fällen nicht den gesamten Buchstaben, sondern konzentrieren sich in etwa auf den Bereich, in dem sich der Strich des Buchstabens *e* kreuzt. Dieses Merkmal ist, den Class Activation Maps nach zu urteilen, gut durch die Faltungsschichten erkennbar und aussagekräftig für die Präsenz des Zeichens *e*.

Weniger eindeutig sind die Heatmaps für das Beispielbild mit dem Zeichen *f*. Zwar wird bei beiden Architekturen das *f* im Wort *life* hervorgehoben, dies gilt jedoch z. B. auch für die Zeichen *P*, *e* und *r* im Wort *Peers* oder das *t* in *nominating*. Die hervor-



Transkription: *Nominating any more Labour life Peers*

Abbildung 5.4.6: Class Activation Maps für die Buchstaben *e*, *f* und *y* zu den angepassten Architekturen von Puigcerver und Wigington et al. Die Heatmaps für das Beispiel aus dem IAM Datensatz [MB02] wurden halbtransparent über das Zeilenabbild gelegt.

gehobenen Bereiche betreffen hier insbesondere die Kreuzung von senkrechten und waagerechten Strichen und den oberen Bogen, die durch die Filter der Faltungsblöcke erkannt wurden. Diese Merkmale scheinen jedoch nicht einzigartig für das Zeichen *f* zu sein.

Für viele der nicht gezeigten Zeichen sind die zugehörigen Heatmaps vergleichbar mit dem Beispiel zum Buchstaben *y*. Nahezu alle beschriebenen Bildbereiche werden hervorgehoben und es ist kaum eine Korrelation zu den tatsächlichen Vorkommen des Zeichens *y* erkennbar. Lediglich bei der Architektur von Wigington et al. scheinen Merkmale gelernt worden zu sein, die eine schwache Korrelation mit den Unterlängen der Zeichen *g* und *y* zeigen. Eine Lokalisierung ist allein durch diese Heatmaps, unabhängig von der Architektur, aber nicht hinreichend möglich. Im Class Activation Ansatz hat die erheblich größere Anzahl von Faltungsmerkmalen der Wigington Architektur keinen offensichtlich erkennbaren, positiven Einfluss bei der Lokalisierung von Buchstaben.

Der Informationsgehalt der Class Activation Maps lässt den Schluss zu, dass der Großteil der eigentlichen Erkennung und Transkription der Zeichen innerhalb der rekurrenten Schichten stattfindet. Die Faltungsschichten können Merkmale zu verschiedenen charakteristischen Teilen von Zeichen berechnen, aber scheinen in den vorliegenden Architekturen nicht dazu in der Lage zu sein, die Gestalt ganzer Zeichen zu erfassen. Den rekurrenten Blöcken kommt demnach nicht nur die Aufgabe zu den zeitlichen Ablauf erkannter Zeichen zu bestimmen und Ambiguitäten aufzulösen, sondern auch die eigentliche Erkennung von Zeichen durch den räumlichen Zusammenhang ihrer Merkmale aus den Faltungsschichten abzubilden.

6 FAZIT

In den vorherigen Kapiteln der vorliegenden Arbeit wurde die Leistungsfähigkeit verschiedener Ansätze für die Transkription handschriftlich erstellter Texte unter der Verwendung von Convolutional Recurrent Neural Networks und der Connectionist Temporal Classification untersucht. Dazu wurden die beiden Netzarchitekturen von Joan Puigcerver und Wigington et al. sowie einige Abwandlungen davon auf den drei Datensätzen IAM Handwriting Database, RIMES Database und George Washington Database evaluiert, um den Effekt verschiedener Parametrierungen der Architekturen zu analysieren. Die betrachteten Architekturen unterscheiden sich grundlegend in der Anzahl der berechneten Merkmale innerhalb der Faltungsschichten sowie in der Tiefe des rekurrenten Teils, was zu Differenzen in der Leistungsfähigkeit innerhalb der zeilen- und wortbasierten Transkription auf den verschiedenen Datensätzen führt.

Während die Architektur von Puigcerver mithilfe ihrer fünf LSTM-Schichten besonders niedrige Zeichen- und Wortfehlerraten innerhalb der zeilenbasierten Transkription auf den IAM Handwriting und RIMES Datensätzen zeigte, konnten die besten Ergebnisse für die wortbasierte Transkription zumindest auf den RIMES und George Washington Datensätzen mit der Architektur von Wigington et al. erzielt werden. Der Vorteil des tiefen rekurrenten Teils der CRNN-Architektur von Puigcerver zeigt sich insbesondere bei der Analyse von langen Sequenzen, wie sie in der zeilenbasierten Transkription vorkommen. Auch durch die qualitative Bewertung der Class Activation Maps wird die große Bedeutung der rekurrenten Blöcke für die Transkription von handschriftlich erstellten Texten mit der CRNN-Architektur sichtbar.

Der Einsatz der affinen oder Random Grid Warp Distortion Augmentierungsoperationen zur Begrenzung der Überanpassung der Neuronalen Netze führte unabhängig vom Datensatz und der verwendeten Netzarchitektur zur erheblichen Verringerung der Fehlerraten. Insbesondere bei der zeilenbasierten Transkription konnten mit der Random Grid Warp Distortion von Wigington et al. die besten Ergebnisse erzielt werden. Im wortbasierten Fall waren die Zeichen- und Wortfehlerraten beim Einsatz der affinen Augmentierung meist besser. Die Signifikanz dieser eher geringen Unterschiede zwischen den beiden Augmentierungsverfahren konnte durch statistische Tests jedoch nicht nachgewiesen werden.

Beim Vergleich der Ergebnisse zwischen verschiedenen Verfahren sollte die Wahl der Hyperparameter wie dem verwendeten Alphabet, Einteilung des Datensatzes, Umfang

der Trainingsdaten, Normalisierung der Annotation, Vorverarbeitung der Eingaben oder der Einsatz eines externen Sprachmodells sorgfältig überprüft werden. Es ist nicht immer offensichtlich erkennbar, welche Hyperparemeter und Techniken (z. B. Test-Side Augmentation) für die Evaluation eines Verfahrens auf den verschiedenen Datensätzen eingesetzt wurden. Dies ist jedoch notwendig, um einen fairen Vergleich sicherzustellen und eine Leistungsdifferenz z. B. dem Einsatz einer neuen Augmentierungsoperation oder einer verbesserten Netzarchitektur gesichert zuschreiben zu können.

Die detaillierte Betrachtung der Fehlerraten auf der Zeichenebene verdeutlicht die Korrelation zwischen der Häufigkeit eines Buchstabens im Trainings-Set und seiner Fehlerrate auf dem Evaluations-Set. Zusammen mit der geringen relativen Häufigkeit vieler Großbuchstaben und auch gängiger Sonderzeichen hebt dies die Bedeutung der Entwicklung größerer Datensätze hervor, um die Leistungsfähigkeit der Verfahren zur Handschrifterkennung dem Niveau moderner OCR Programme für maschinell erstellte Dokumente anzunähern. Der geringe Umfang der Validierungs-Sets verursacht auch die Breite der berechneten Konfidenzintervalle und die fehlende Signifikanz der Leistungsunterschiede zwischen den Verfahren, die sich in vielen Fällen durch die statistischen Tests ergeben hat. Dies erschwert den fundierten Vergleich von Verfahren mit ähnlichen Fehlerraten, verdeutlicht aber die Relevanz von statistischen Analysen der Ergebnisse wissenschaftlicher Arbeiten.

Eine nahezu fehlerfreie Erkennung von Textdokumenten mit uneingeschränkter Handschrift ist auch mit aktuellen Verfahren noch nicht möglich und die Handschrifterkennung bleibt ein wichtiges Forschungsthema. Anhand der Wortfehlerraten ist zu erkennen, dass die in dieser Arbeit vorgestellten Verfahren noch ca. jedes fünfte Wort falsch transkribieren. Eine Möglichkeit zur Verbesserung bestehender Systeme ist die Erzeugung synthetischer Daten für das Training. Verfahren wie *CPWN* von Shen und Messina [SM16] können ganze Dokumentseiten auf der Basis eines auf Buchstabenebene segmentierten Datensatzes erzeugen. Mit Techniken wie dieser können verfügbare Datensätze mit vertretbarem Aufwand erheblich vergrößert und der Einsatz tieferer Netzarchitekturen mit einer größeren Anzahl von Parametern ermöglicht werden. Andere Methoden wie AdaBoost [FS97] haben das Potenzial, den Umgang mit den unterschiedlichen relativen Häufigkeiten zwischen den Zeichen des Alphabets zu verbessern. Die Erweiterung auf die Transkription ganzer Paragraphen anstatt der wort- oder zeilenbasierten Transkription, z. B. durch Neuronale Netze mit Aufmerksamkeitsmechanismus, ist ein Schritt hin zur automatischen Analyse des Inhalts ganzer Dokumentenseiten. Dies ist nur eine Auswahl von interessanten Möglichkeiten zur Weiterentwicklung der vorgestellten Methodik.

LITERATURVERZEICHNIS

[AAB+15] Abadi, Martín ; Agarwal, Ashish ; Barham, Paul ; Brevdo, Eugene ; Chen, Zhifeng ; Citro, Craig ; Andy Davis, Greg S. C. ; Dean, Jeffrey ; Devin, Matthieu ; Ghemawat, Sanjay ; Goodfellow, Ian ; Harp, Andrew ; Irving, Geoffrey ; Isard, Michael ; Jia, Yangqing ; Jozefowicz, Rafal ; Kaiser, Lukasz ; Kudlur, Manjunath ; Levenberg, Josh u. a.: *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. https://www.tensorflow.org/. Version: 2015. – Software available from tensorflow.org

[ACG+06] Augustin, Emmanuel ; Carré, Matthieu ; Grosicki, Emmanuèle ; Brodin, J.-M. ; Geoffrois, Edouard ; Preteux, Francoise: RIMES evaluation campaign for handwritten mail processing. In: *International Workshop on Frontiers in Handwriting Recognition (IWFHR'06),*, 2006, 231-235

[AGFV14] Almazán, Jon ; Gordo, Albert ; Fornés, Alicia ; Valveny, Ernest: Word spotting and recognition with embedded attributes. In: *IEEE transactions on pattern analysis and machine intelligence* 36 (2014), Nr. 12, S. 2552–2566

[BCB14] Bahdanau, Dzmitry ; Cho, Kyunghyun ; Bengio, Yoshua: Neural machine translation by jointly learning to align and translate. In: *arXiv preprint arXiv:1409.0473* (2014). – ICLR 2015 Presentation

[Bil17] Bilof, Randall (Hrsg.): *14th IAPR International Conference on Document Analysis and Recognition, ICDAR*. Kyoto, Japan : IEEE Computer Society, 2017 . – ISBN 978–1–5386–3586–5

[Bis06] Bishop, Christopher: *Pattern recognition and machine learning*. Springer-Verlag New York, 2006

[BLCW09] Bengio, Yoshua ; Louradour, Jérôme ; Collobert, Ronan ; Weston, Jason: Curriculum learning. In: *Proceedings of the 26th annual international conference on machine learning* ACM, 2009, S. 41–48

[BLM17] Bluche, Théodore ; Louradour, Jérôome ; Messina, Ronaldo: Scan, attend and read: End-to-end handwritten paragraph recognition with

mdlstm attention. In: *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)* Bd. 1 IEEE, 2017, S. 1050–1055

[BLNB95] Bengio, Yoshua ; LeCun, Yann ; Nohl, Craig ; Burges, Chris: LeRec: A NN/HMM hybrid for on-line handwriting recognition. In: *Neural Computation* 7 (1995), Nr. 6, S. 1289–1303

[Blu15] Bluche, Théodore: *Deep neural networks for large vocabulary handwritten text recognition*, Université Paris Sud-Paris XI, Diss., 2015

[BM12] Bourlard, Herve A. ; Morgan, Nelson: *Connectionist speech recognition: a hybrid approach*. Bd. 247. Springer Science & Business Media, 2012

[Chr15a] Christopher, Olah: *Neural Networks, Types, and Functional Programming*. "http://colah.github.io/posts/2015-09-NN-Types-FP/". Version: 2015. – Zugriff: 19.05.2019

[Chr15b] Christopher, Olah: *Understanding LSTM Networks*. "https://colah.github.io/posts/2015-08-Understanding-LSTMs/". Version: 2015. – Zugriff: 05.05.2019

[CIR12] Cho, KyungHyun ; Ilin, Alexander ; Raiko, Tapani: Tikhonov-Type Regularization for Restricted Boltzmann Machines. In: Villa, Alessandro E. P. (Hrsg.) ; Duch, Włodzisław (Hrsg.) ; Érdi, Péter (Hrsg.) ; Masulli, Francesco (Hrsg.) ; Palm, Günther (Hrsg.): *Artificial Neural Networks and Machine Learning – ICANN 2012*. Berlin, Heidelberg : Springer, 2012. – ISBN 978–3–642–33269–2, S. 81–88

[CMG+14] Cho, Kyunghyun ; Merriënboer, Bart van ; Gulcehre, Caglar ; Bahdanau, Dzmitry ; Bougares, Fethi ; Schwenk, Holger ; Bengio, Yoshua: Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar : Association for Computational Linguistics, 2014, S. 1724–1734

[CMS12] Cireşan, D. ; Meier, U. ; Schmidhuber, J.: Multi-column deep neural networks for image classification. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012. – ISSN 1063–6919, S. 3642–3649

[CUH15] Clevert, Djork-Arné ; Unterthiner, Thomas ; Hochreiter, Sepp: Fast and accurate deep network learning by exponential linear units (elus). In: *arXiv preprint arXiv:1511.07289* (2015). – Poster Presentation ICLR 2016

[DHS11] Duchi, John ; Hazan, Elad ; Singer, Yoram: Adaptive subgradient methods for online learning and stochastic optimization. In: *Journal of Machine Learning Research* 12 (2011), Nr. Jul, S. 2121–2159

[DKLM05] Dekking, Frederik M. ; Kraaikamp, Cornelis ; Lopuhaä, Hendrik P. ; Meester, Ludolf E.: *A Modern Introduction to Probability and Statistics: Understanding why and how*. Springer Science & Business Media, 2005

[DT05] Dalal, Navneet ; Triggs, Bill: Histograms of oriented gradients for human detection. In: *2005 International Conference on Computer Vision & Pattern Recognition (CVPR)* Bd. 1 IEEE Computer Society, 2005, S. 886–893

[Fin14] Fink, Gernot A.: *Advances in Computer Vision and Pattern Recognition*. Bd. 2: *Markov Models for Pattern Recognition: From Theory to Applications*. London : Springer, 2014

[FKFB12] Fischer, A ; Keller, A ; Frinken, V ; Bunke, H: Lexicon-Free Handwritten Word Spotting Using Character HMMs,. In: *Pattern Recognition Letters* 33 (2012), Nr. 7, S. 934–942

[FS97] Freund, Yoav ; Schapire, Robert E.: A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. In: *Journal of Computer and System Sciences* 55 (1997), Nr. 1, S. 119 – 139. – ISSN 0022–0000

[GB10] Glorot, Xavier ; Bengio, Yoshua: Understanding the difficulty of training deep feedforward neural networks. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, S. 249–256

[GBC16] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning*. MIT Press, 2016. – http://www.deeplearningbook.org

[GEA09] Grosicki, Emmanuèle ; El Abed, Haikal: ICDAR 2009 handwriting recognition competition. In: *2009 10th International Conference on Document Analysis and Recognition (ICDAR)* IEEE, 2009, S. 1398–1402

[GFGS06] Graves, Alex ; Fernández, Santiago ; Gomez, Faustino ; Schmidhuber, Jürgen: Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In: *Proceedings of the 23rd International Conference on Machine Learning* ACM, 2006, S. 369–376

[GFS07] Graves, Alex ; Fernández, Santiago ; Schmidhuber, Jürgen: Multi-dimensional Recurrent Neural Networks. In: Sá, Joaquim M. (Hrsg.) ; Alexandre, Luís A. (Hrsg.) ; Duch, Włodzisław (Hrsg.) ; Mandic, Danilo (Hrsg.): *Artificial Neural Networks – ICANN 2007*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2007. – ISBN 978–3–540–74690–4, S. 549–558

[GLF+93] Garofolo, J. S. ; Lamel, L. F. ; Fisher, W. M. ; Fiscus, J. G. ; Pallett, D. S. ; Dahlgren, N. L.: *TIMIT Acoustic-Phonetic Continuous Speech Corpus*. 1993

[GLF+08] Graves, Alex ; Liwicki, Marcus ; Fernández, Santiago ; Bertolami, Roman ; Bunke, Horst ; Schmidhuber, Jürgen: A Novel Connectionist System for Unconstrained Handwriting Recognition. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31 (2008), Nr. 5, S. 855–868

[Gra08] Graves, Alex: *Supervised Sequence Labelling with Recurrent Neural Networks*, Technische Universität München, Dissertation, 2008. https://www.cs.toronto.edu/~graves/. – 20–26 S.

[GS00] Gers, Felix A. ; Schmidhuber, Jürgen: Recurrent nets that time and count. In: *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium* Bd. 3, 2000. – ISSN 1098–7576, S. 189–194 vol.3

[GS09] Graves, Alex ; Schmidhuber, Jürgen: Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks. In: Koller, D. (Hrsg.) ; Schuurmans, D. (Hrsg.) ; Bengio, Y. (Hrsg.) ; Bottou, L. (Hrsg.): *Advances in Neural Information Processing Systems 21*. Curran Associates, Inc., 2009, S. 545–552

[GSC00] Gers, Felix A. ; Schmidhuber, Jürgen ; Cummins, Fred: Learning to Forget: Continual Prediction with LSTM. In: *Neural Computation* 12 (2000), Nr. 10, S. 2451–2471

[Gue18] Guerrero, Juan E. (Hrsg.): *2018 16th International Conference on Frontiers in Handwriting Recognition, ICFHR*. Niagara Falls, NY, USA : IEEE Computer Society, 2018 . – ISBN 978–1–5386–5875–8

[HK12] Harada, Tatsuya ; Kuniyoshi, Yasuo: Graphical Gaussian Vector for Image Categorization. In: Pereira, F. (Hrsg.) ; Burges, C. J. C. (Hrsg.)

; BOTTOU, L. (Hrsg.) ; WEINBERGER, K. Q. (Hrsg.): *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, S. 1547–1555

[HS97] HOCHREITER, Sepp ; SCHMIDHUBER, Jürgen: Long Short-Term Memory. In: *Neural Computation* 9 (1997), Nr. 8, S. 1735–1780

[HSK+12] HINTON, Geoffrey E. ; SRIVASTAVA, Nitish ; KRIZHEVSKY, Alex ; SUTSKEVER, Ilya ; SALAKHUTDINOV, Ruslan R.: Improving neural networks by preventing co-adaptation of feature detectors. In: *arXiv preprint arXiv:1207.0580* (2012)

[HSW89] HORNIK, Kurt ; STINCHCOMBE, Maxwell ; WHITE, Halbert: Multilayer feedforward networks are universal approximators. In: *Neural Networks* 2 (1989), Nr. 5, S. 359 – 366. – ISSN 0893–6080

[HZRS15] HE, Kaiming ; ZHANG, Xiangyu ; REN, Shaoqing ; SUN, Jian: Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37 (2015), Nr. 9, S. 1904–1916

[HZRS16] HE, Kaiming ; ZHANG, Xiangyu ; REN, Shaoqing ; SUN, Jian: Deep Residual Learning for Image Recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, S. 770–778

[IS15] IOFFE, Sergey ; SZEGEDY, Christian: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: BACH, Francis (Hrsg.) ; BLEI, David (Hrsg.): *Proceedings of the 32nd International Conference on Machine Learning* Bd. 37. Lille, France : PMLR, 2015 (Proceedings of Machine Learning Research), S. 448–456

[KB14] KINGMA, Diederik P. ; BA, Jimmy: Adam: A method for stochastic optimization. In: *arXiv preprint arXiv:1412.6980* (2014). – Poster Präsentation ICLR 2015

[KHB88] KUNDU, A. ; HE, Y. ; BAHL, P.: Recognition of handwritten word: first and second order hidden Markov model based approach. In: *Proceedings CVPR '88: The Computer Society Conference on Computer Vision and Pattern Recognition*, 1988. – ISSN 1063–6919, S. 457–462

[KMB+10] KERMORVANT, Christopher ; MENASRI, Farès ; BIANNE, A-L ; AL-HAJJ, R ; MOKBEL, Chafic ; LIKFORMAN-SULEM, Laurence: The A2iA-Telecom

ParisTech-UOB system for the ICDAR 2009 handwriting recognition competition. In: *2010 12th International Conference on Frontiers in Handwriting Recognition* IEEE, 2010, S. 247–252

[KSH12] Krizhevsky, Alex ; Sutskever, Ilya ; Hinton, Geoffrey E.: ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira, F. (Hrsg.) ; Burges, C. J. C. (Hrsg.) ; Bottou, L. (Hrsg.) ; Weinberger, K. Q. (Hrsg.): *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, S. 1097–1105

[LB05] Liwicki, Marcus ; Bunke, Horst: IAM-OnDB-an on-line English sentence database acquired from handwritten text on a whiteboard. In: *2005 Eighth International Conference on Document Analysis and Recognition (ICDAR)* IEEE, 2005, S. 956–961

[LBBH98] Lecun, Yann ; Bottou, Léon ; Bengio, Yoshua ; Haffner, Patrick: Gradient-based learning applied to document recognition. In: *Proceedings of the IEEE* 86 (1998), Nr. 11, S. 2278–2324

[LBOM98] *Kapitel* Speeding Learning. In: LeCun, Yann ; Bottou, Leon ; Orr, Genevieve B. ; Müller, Klaus R.: *Efficient BackProp*. Berlin, Heidelberg : Springer, 1998. – ISBN 978–3–540–49430–0, S. 9–50

[Lev66] Levenshtein, Vladimir I.: Binary codes capable of correcting deletions, insertions, and reversals. In: *Soviet physics doklady* Bd. 10, 1966, S. 707–710

[LSD15] Long, Jonathan ; Shelhamer, Evan ; Darrell, Trevor: Fully Convolutional Networks for Semantic Segmentation. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, S. 3431–3440

[MA07] Margner, Volker ; Abed, Haikal E.: Arabic Handwriting Recognition Competition. In: *2007 Ninth International Conference on Document Analysis and Recognition (ICDAR)* Bd. 2, 2007. – ISSN 1520–5363, S. 1274–1278

[MB02] Marti, Urs-Viktor ; Bunke, Horst: The IAM-database: an English sentence database for offline handwriting recognition. In: *International Journal on Document Analysis and Recognition* 5 (2002), Nr. 1, S. 39–46

[MHN13] Maas, Andrew L. ; Hannun, Awni Y. ; Ng, Andrew Y.: Rectifier Nonlinearities Improve Neural Network Acoustic Models. In: *2013 ICML Workshop on Deep Learning for Audio, Speech, and Language Processing (WDLASL)*, 2013

[MM19] Moysset, Bastien ; Messina, Ronaldo: Are 2D-LSTM really dead for offline text recognition? In: *International Journal on Document Analysis and Recognition (IJDAR)* (2019), Jun

[MP43] McCulloch, Warren S. ; Pitts, Walter: A Logical Calculus of the Ideas Immanent in Nervous Activity. In: *The Bulletin of Mathematical Biophysics* 5 (1943), Nr. 4, S. 115–133. – ISSN 1522–9602

[NH10] Nair, Vinod ; Hinton, Geoffrey E.: Rectified Linear Units Improve Restricted Boltzmann Machines. In: *2010 Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010, S. 807–814

[Nie03] Niemann, Heinrich: *Klassifikation von Mustern*. Universität Erlangen, 2003 http://www5.informatik.uni-erlangen.de/fileadmin/Persons/NiemannHeinrich/klassifikation-von-mustern/m00-www.pdf. – 2. Auflage

[NWF86] Nag, R. ; Wong, K. ; Fallside, F.: Script Recognition using Hidden Markov Models. In: *ICASSP '86. IEEE International Conference on Acoustics, Speech, and Signal Processing* Bd. 11, 1986, S. 2071–2074

[PBKL14] Pham, Vu ; Bluche, Théodore ; Kermorvant, Christopher ; Louradour, Jérôme: Dropout Improves Recurrent Neural Networks for Handwriting Recognition. In: *2014 14th International Conference on Frontiers in Handwriting Recognition (ICFHR)* IEEE, 2014, S. 285–290

[PGC+17] Paszke, Adam ; Gross, Sam ; Chintala, Soumith ; Chanan, Gregory ; Yang, Edward ; DeVito, Zachary ; Lin, Zeming ; Desmaison, Alban ; Antiga, Luca ; Lerer, Adam: Automatic differentiation in PyTorch. (2017)

[PMB13] Pascanu, Razvan ; Mikolov, Tomas ; Bengio, Yoshua: On the difficulty of training recurrent neural networks. In: Dasgupta, Sanjoy (Hrsg.) ; McAllester, David (Hrsg.): *Proceedings of the 30th International Conference on Machine Learning* Bd. 28. Atlanta, Georgia, USA : PMLR, 2013 (Proceedings of Machine Learning Research 3), S. 1310–1318

[PMM+02] Pechwitz, Mario ; Maddouri, S S. ; Märgner, Volker ; Ellouze, Noureddine ; Amiri, Hamid: IFN/ENIT-database of handwritten Arabic words. In: *Proceedings of 7th Colloque International Francophone sur l'Ecrit et le Document (CIFED)* Bd. 2. Hammamet, Tunis, 2002, S. 127–136

[Pre12] *Kapitel* Regularization Techniques to Improve Generalization. In: PRECHELT, Lutz: *Early Stopping — But When?* Berlin, Heidelberg : Springer Berlin Heidelberg, 2012. – ISBN 978–3–642–35289–8, S. 53–67

[Pui17] PUIGCERVER, Joan: Are Multidimensional Recurrent Layers Really Necessary for Handwritten Text Recognition? In: *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)* Bd. 1 IEEE, 2017, S. 67–72

[PW16] POZNANSKI, Arik ; WOLF, Lior: CNN-N-Gram for HandwritingWord Recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. – ISSN 1063–6919, S. 2305–2314

[RDS+15] RUSSAKOVSKY, Olga ; DENG, Jia ; SU, Hao ; KRAUSE, Jonathan ; SATHEESH, Sanjeev ; MA, Sean ; HUANG, Zhiheng ; KARPATHY, Andrej ; KHOSLA, Aditya ; BERNSTEIN, Michael ; BERG, Alexander C. ; FEI-FEI, Li: ImageNet Large Scale Visual Recognition Challenge. In: *International Journal of Computer Vision* 115 (2015), Nr. 3, S. 211–252. – ISSN 1573–1405

[RHGS15] REN, Shaoqing ; HE, Kaiming ; GIRSHICK, Ross ; SUN, Jian: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In: CORTES, Corinna (Hrsg.) ; LAWRENCE, Neil D. (Hrsg.) ; LEE, Daniel D. (Hrsg.) ; SUGIYAMA, Masashi (Hrsg.) ; GARNETT, Roman (Hrsg.): *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2015, S. 91–99

[RHW88] RUMELHART, David E. ; HINTON, Geoffrey E. ; WILLIAMS, Ronald J.: Learning representations by back-propagating errors. In: *Cognitive modeling* 5 (1988), Nr. 3, S. 1

[Roj96] ROJAS, Raúl: *Neural Networks: A Systematic Introduction*. Berlin, New-York : Springer Verlag, 1996

[SAC07] SMUCKER, Mark D. ; ALLAN, James ; CARTERETTE, Ben: A Comparison of Statistical Significance Tests for Information Retrieval Evaluation. In: *Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management*. Lissabon, Portugal : ACM, 2007, S. 623–632

[Say73] SAYRE, Kenneth M.: Machine recognition of handwritten words: A project report. In: *Pattern Recognition* 5 (1973), Nr. 3, S. 213 – 228. – ISSN 0031–3203

[SBY17] Shi, Baoguang ; Bai, Xiang ; Yao, Cong: An End-to-End Trainable Neural Network for Image-based Sequence Recognition and its Application to Scene Text Recognition. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39 (2017), Nr. 11, S. 2298–2304

[Sch18] Scheidl, Harald: *An Intuitive Explanation of Connectionist Temporal Classification*. https://towardsdatascience.com/intuitively-understanding-connectionist-temporal-classification-3797e43a86c/. Version: 2018. – Zugriff: 15.07.2019

[SF16] Sudholt, Sebastian ; Fink, Gernot A.: PHOCNet: A Deep Convolutional Neural Network for Word Spotting in Handwritten Documents. In: *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, 2016. – ISSN 2167–6445, S. 277–282

[SGH95] Schenkel, Markus ; Guyon, Isabelle ; Henderson, Donnie: On-line cursive script recognition using time-delay neural networks and hidden Markov models. In: *Machine Vision and Applications* 8 (1995), Nr. 4, S. 215–223

[SHK+14] Srivastava, Nitish ; Hinton, Geoffrey ; Krizhevsky, Alex ; Sutskever, Ilya ; Salakhutdinov, Ruslan: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. In: *The Journal of Machine Learning Research* 15 (2014), Nr. 1, S. 1929–1958

[SLJ+15] Szegedy, Christian ; Liu, Wei ; Jia, Yangqing ; Sermanet, Pierre ; Reed, Scott ; Anguelov, Dragomir ; Erhan, Dumitru ; Vanhoucke, Vincent ; Rabinovich, Andrew: Going Deeper with Convolutions. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. – ISSN 1063–6919, S. 1–9

[SM16] Shen, Xi ; Messina, Ronaldo: A Method of Synthesizing Handwritten Chinese Images for Data Augmentation. In: *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, 2016. – ISSN 2167–6445, S. 114–119

[SMB10] Scherer, Dominik ; Müller, Andreas ; Behnke, Sven: Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition. In: Diamantaras, Konstantinos (Hrsg.) ; Duch, Wlodek (Hrsg.) ; Iliadis, Lazaros S. (Hrsg.): *2010 Proceedings of the 20th International Conference on*

Artificial Neural Networks (ICANN). Berlin, Heidelberg : Springer Berlin Heidelberg, 2010. – ISBN 978–3–642–15825–4, S. 92–101

[SMSC94] Starner, Thad ; Makhoul, John ; Schwartz, Richard ; Chou, George: Online cursive handwriting recognition using speech recognition methods. In: *1994 Proceedings the International Conference on Acoustics, Speech and Signal Processing (ICASSP* IEEE, 1994, S. V–125

[SP97] Schuster, Mike ; Paliwal, Kuldip K.: Bidirectional Recurrent Neural Networks. In: *IEEE Transactions on Signal Processing* 45 (1997), Nr. 11, S. 2673–2681. – ISSN 1053–587X

[SRTV16] Sánchez, Joan A. ; Romero, Verónica ; Toselli, Alejandro H. ; Vidal, Enrique: ICFHR2016 Competition on Handwritten Text Recognition on the READ Dataset. In: *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, 2016. – ISSN 2167–6445, S. 630–635

[SSP03] Simard, Patrice Y. ; Steinkraus, Dave ; Platt, John C.: Best practices for convolutional neural networks applied to visual document analysis. In: *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, 2003, S. 958–963

[Sud18] Sudholt, Sebastian: *Learning Attribute Representations with Deep Convolutional Neural Networks for Word Spotting*, Technische Universität Dortmund, Diss., 2018

[SZ14] Simonyan, Karen ; Zisserman, Andrew: Very Deep Convolutional Networks for Large-Scale Image Recognition. In: *arXiv e-print arXiv:1409.1556* (2014). – Präsentation ICLR 2015

[TGJ+15] Tompson, Jonathan ; Goroshin, Ross ; Jain, Arjun ; LeCun, Yann ; Bregler, Christoph: Efficient Object Localization Using Convolutional Networks. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, S. 648–656

[TH12] Tieleman, Tijmen ; Hinton, Geoffrey: Lecture 6.5-rmsprop: Divide the Gradient by a Running Average of its Recent Magnitude. In: *COURSERA: Neural Networks for Machine Learning* 4 (2012), Nr. 2, S. 26–31

[TPMEA14] Tong, Audrey ; Przybocki, Mark ; Märgner, Volker ; El Abed, Haikal: NIST 2013 Open Handwriting Recognition and Translation (Open

HaRT'13) Evaluation. In: *2014 11th IAPR International Workshop on Document Analysis Systems* IEEE, 2014, S. 81–85

[VDW+15] Voigtlaender, Paul ; Doetsch, Patrick ; Wiesler, Simon ; Schlüter, Ralf ; Ney, Hermann: Sequence-discriminative training of recurrent neural networks. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* IEEE, 2015, S. 2100–2104

[VRS15] Villegas, Mauricio ; Romero, Verónica ; Sánchez, Joan A.: On the Modification of Binarization Algorithms to Retain Grayscale Information for Handwritten Text Recognition. In: *IbPRIA 2015: Iberian Conference on Pattern Recognition and Image Analysis* Springer, 2015, S. 208–215

[WFS05] Wienecke, Markus ; Fink, Gernot A. ; Sagerer, Gerhard: Toward automatic video-based whiteboard reading. In: *International Journal of Document Analysis and Recognition (IJDAR)* 7 (2005), Nr. 2, S. 188–200. – ISSN 1433–2825

[WSD+17] Wigington, Curtis ; Stewart, Seth ; Davis, Brian ; Barrett, Bill ; Price, Brian ; Cohen, Scott: Data Augmentation for Recognition of Handwritten Words and Lines Using a CNN-LSTM Network. In: *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)* Bd. 1 IEEE, 2017, S. 639–645

[ZKL+14] Zhou, Bolei ; Khosla, Aditya ; Lapedriza, Agata ; Oliva, Aude ; Torralba, Antonio: Object detectors emerge in Deep Scene CNNs. In: *arXiv preprint arXiv:1412.6856* (2014). – ICLR 2015 Presentation

[ZKL+16] Zhou, B. ; Khosla, A. ; Lapedriza, A. ; Oliva, A. ; Torralba, A.: Learning Deep Features for Discriminative Localization. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. – ISSN 1063–6919, S. 2921–2929

[ZLK+17] Zhou, Bolei ; Lapedriza, Agata ; Khosla, Aditya ; Oliva, Aude ; Torralba, Antonio: Places: A 10 million Image Database for Scene Recognition. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2017)

ABBILDUNGSVERZEICHNIS

Abbildung 1.0.1 Darstellung der seiten-, zeilen- und wortbasierten Transkription anhand eines Beispiels aus der IAM Database [MB02]. 5
Abbildung 2.1.1 Pipeline eines klassischen Mustererkennungssystems und eines Systems mit Neuronalen Netzen. 8
Abbildung 2.2.1 Abstraktes Neuron nach [Roj96]. 10
Abbildung 2.2.2 Visualisierung des Trainingsprozesses mit zu großer und zu kleiner Lernrate. 12
Abbildung 2.3.1 Visualisierung einer Faltungsschicht nach [Sud18]. 16
Abbildung 2.4.1 Gegenüberstellung der Trainings- und Validierungs-Kosten je Epoche aus [GBC16]. 19
Abbildung 2.4.2 Visualisierung von Dropout nach [SHK+14] 20
Abbildung 2.4.3 Visualisierung verschiedener Aktivierungsfunktionen. 22
Abbildung 2.5.1 Darstellung einer Zelle eines rekurrenten Neuronalen Netzes nach [Chr15b]. 24
Abbildung 2.5.2 Entfaltetes rekurrentes Netz für die Backpropagation Through Time nach [Chr15b]. 25
Abbildung 2.5.3 Visualisierung einer LSTM Zelle nach [Chr15b]. 27
Abbildung 2.5.4 Visualisierung einer GRU Zelle nach [Chr15b]. 29
Abbildung 2.5.5 Entfaltetes bidirektionales rekurrentes Netz nach [Chr15a]. 30
Abbildung 2.5.6 Visualisierung eines zweidimensionalen rekurrenten Neuronalen Netzes nach [GFS07]. 31
Abbildung 2.6.1 Visualisierung des Best Path Decodings der Connectionist Temporal Classification nach [Sch18]. 36
Abbildung 3.1.1 Darstellung der LeNet-Architektur aus [LBBH98]. 38
Abbildung 3.1.2 Darstellung der Architektur des AlexNet aus [KSH12]. 39
Abbildung 3.1.3 Visualisierung der Faltungsnetzarchitektur VGG16E von [SZ14]. 40
Abbildung 3.2.1 Schema der OUTSEG-Architektur aus [BLNB95]. 41
Abbildung 3.3.1 Visualisierung der Pyramidal Histogram of Characters aus [SF16]. 43
Abbildung 3.3.2 Visualisierung der CNN-N-Gramm Architektur aus [PW16]. 44
Abbildung 3.3.3 Darstellung der PHOCNet Architektur aus [SF16]. 45
Abbildung 3.4.1 Übersicht über die blockbasierte Connectionist Temporal Classification Architektur aus [GS09]. 47

Abbildung 3.5.1 Visualisierung der Aufmerksamkeits-basierten MDLSTM Architektur aus [BLM17]. 49
Abbildung 3.6.1 Anwendung affiner Transformationen auf ein Wortabbild. 51
Abbildung 3.6.2 Visualisierung der elastischen Deformation aus [SSP03]. 51
Abbildung 3.6.3 Visualisierung der Random Grid Warp Augmentation von [WSD+17]. 52
Abbildung 3.7.1 Übersicht über das Class Activation Mapping Verfahren aus [ZKL+16]. 53
Abbildung 4.1.1 Überblick über die CRNN-Architektur aus [SBY17]. 56
Abbildung 4.1.2 Visualisierung zur Erzeugung der Mermalssequenz für die rekurrenten Schichten aus [SBY17]. 57
Abbildung 4.2.1 Visualisierung der CRNN-Architektur von Joan Puigcerver [Pui17]. 59
Abbildung 4.2.2 Gegenüberstellung der Merkmale einer 2D-LSTM und Faltungsschicht aus [Pui17]. 60
Abbildung 4.3.1 Visualisierung der CRNN-Architektur von Wigington et al. [WSD+17]. 61
Abbildung 4.4.1 Schema des Class Activation Mappings für die Handschrifterkennung auf der Basis von Convolutional Recurrent Neural Networks. 62
Abbildung 5.1.1 Beispiel für eine Zeile aus der IAM Handwriting Database [MB02]. 64
Abbildung 5.1.2 Beispiel für eine Zeile aus der RIMES Database [ACG+06]. 65
Abbildung 5.1.3 Ein Auszug aus der George Washington Database [FKFB12]. 66
Abbildung 5.4.1 Verlauf des Durchschnitts der CTC Kosten und der Zeichenfehlerrate in jeder Trainingsepoche. 74
Abbildung 5.4.2 Gegenüberstellung der Häufigkeit von Kleinbuchstaben zu ihrer Fehlerrate. 76
Abbildung 5.4.3 Absolute und relative Anzahl der Vertauschungen zwischen Groß- und Kleinbuchstaben. 77
Abbildung 5.4.4 Verlauf der Aktivierungen je Zeitschritt für ein Beispielbild aus dem IAM Datensatz [MB02]. 80
Abbildung 5.4.5 Darstellung der durchschnittlichen Zeichenfehlerrate für alle vorkommenden Wortlängen. 81
Abbildung 5.4.6 Class Activation Maps für die Buchstaben e, f und y zu den angepassten Architekturen von Puigcerver und Wigington. 90