technische universität
dortmund

Semi-Supervised Learning zur Erkennung von handschriftlichen mathematischen Formeln

Masterarbeit

Marc Ahlers
18. Mai 2023

Supervisors:
Prof. Dr.-Ing. Gernot A. Fink
Fabian Wolf, M.Sc.

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

INHALTSVERZEICHNIS

1 EINLEITUNG

Die Transkription von Freihandeingaben ist ein wichtiger Aspekt bei der fortschreitenden Digitalisierung der Gesellschaft. Mithilfe des Smartphones, Tablets oder Notebooks können schnell und unkompliziert Notizen während der Vorlesung, im Meeting oder beim Telefonat erstellt werden. Dies spielt besonders im Bereich Bildung, Wissenschaft und Technik eine wichtige Rolle, da dort viele handschriftliche Informationen ausgetauscht sowie digital verwendet werden. Jedoch liegen die Freihandeingaben zunächst nur als Bild bzw. Zeichentrajektorie und nicht als Transkription des Geschriebenen vor, wodurch ein einfaches Formatieren, Bearbeiten oder Korrigieren nur händisch auf Bildebene oder erst nach (meist manueller) Transkription möglich ist. Vorallem das Durchsuchen der Notizen ist dabei erst nach einer Transkription möglich. Bei großen Datenmengen, wie beispielsweise die Sammlung an Notizen eines gesamten Semesters, ist eine Suche ein zentrales Werkzeug für einen einfachen und schnellen Umgang.

Durch die Komplexität der Notation, Vielfalt an Schreibstilen und -richtungen ist die Transkription jedoch ein schwieriges Problem. Ein ideales Erkennungssystem, welches alle weltweit auftretenden Glyphen in Schrift oder Formeln unabhängig vom Schreibstil und Layout erkennen kann, erscheint daher utopisch. Selbst bei einem verringerten Problemkreis, beispielsweise der Eingrenzung des Alphabets oder der Sprache, erreichen derzeitige Ansätze im Bereich der *Handwritten Text Recognition* Fehlerraten von rund $4\% - 15\%$ auf Wortebene [TM22, KDHN14].
Bei der Erkennung von mathematischen Formeln ist die Sprache meist zweitrangig, jedoch umfasst das auftretende Alphabet neben lateinischen Schriftzeichen auch griechische Buchstaben oder mathematische Operatoren. Hinzu kommen Notationen wie Tief- bzw. Hochstellung oder Brüche. Bei der Formelerkennung kann daher das Layout der zu erkennenden Zeichen deutlich komplexer werden als bei geschriebenen Sätzen. Hier erreichen derzeitige Erkennungssysteme eine Fehlerrate von rund 35% bezogen auf das fehlerfreie Erkennen einer gesamten Formel [YLD+22].

Die Mehrheit an Erkennungssystemen auf Grundlage von Neuronalen Netzen benötigen möglichst viele Beispieldaten inklusive Annotationen um das Erkennungsproblem besser als zufälliges Raten zu lösen. Manche Architekturen und Problemdomänen

skalieren dabei jedoch besser mit der Menge an verfügbaren Daten. In jedem Fall müssen ebendiese erhoben und gegebenenfalls digitalisiert oder vorverarbeitet werden. Zusätzlich sind Annotationen für eine Teilmenge der Daten für *Supervised Learning* und *Semi-Supervised Learning* zwingend notwendig. Beide Schritte sind jedoch mit Kosten und Zeit verbunden, welche bei einer manuellen Durchführung deutlich größer sind. Um die Kosten zu verringern, wird beim Semi-Supervised Learning eine große Menge nicht-annotierter zusammen mit einer kleinen Menge annotierter Daten verwendet. Die Erkennungsleistung soll dabei besser sein als ein System, welches nur mit dem kleinen Teil annotierter Daten vollständig-überwacht angelernt wird.

Diese Arbeit untersucht in diesem Kontext zwei zentrale Forschungsfragen: In welchem Maß können nicht-annotierte Daten im Training eines Erkennungssystems für handschriftliche mathematische Formeln genutzt werden und inwieweit kann der benötigte annotierte Datenaufwand reduziert werden.

1.1 RAHMEN DER ARBEIT

Um die Forschungsfragen zu untersuchen, werden zunächst die Grundlagen der Musterkennung, Neuronale Netze und der Erkennungspipeline von Bildern und Zeichensequenzen in Kapitel 2 behandelt. Daran anschließend werden die für die Arbeit relevanten verwandten Arbeiten hinsichtlich der mathematischen Formelerkennung, Semi-Supervised Learning und Kalibrierung von Netzwerken in Kapitel 3 thematisiert. Kapitel 4 beinhaltet darauf aufbauend die konkreten Methoden, die in den Experimenten des Kapitels 5 eingesetzt werden. Die Experimente und Methoden werden dabei evaluiert und mit der Literatur verglichen. Abschließend werden die Ergebnisse der Arbeit in Kapitel 6 zusammengefasst und ein Fazit hinsichtlich der Forschungsfragen gezogen.

1.2 BEITRÄGE ZUR FORSCHUNG

In den 6 Kapiteln werden bestehende Methodiken als Grundlage verwendet und mit neuen Ansätzen erweitert. Zusammengefasst entsteht daraus eine neue und veränderte Methodik, welche deutliche Verbesserungen im Vergleich zur Grundlage der Literatur erzielt. Die hauptsächlichen Beiträge dieser Arbeit zur Forschung lassen sich wie folgt zusammenfassen:

- Übertragung des semi-überwachten Self-Trainings auf die mathematische Formelerkennung

- Bedeutung von Kalibrierungsmethoden für weiterführende Methoden sowie für ein stabileres Filtern anhand eines Schwellenwertes
- Fehlerreduktion und Varianzverbesserung durch die *Partial Labeling Heuristik*
- Bedeutung & Wichtigkeit eines Vortrainings mit synthetischen Daten
- Deutliche Verbesserung der Inferenzgeschwindigkeit mithilfe von Beam-Search, Pruning Methoden und speziell *Constant Pruning*
- Freier Zugang zu der Implementierung der vorgestellten Methoden in [AZG23]

1.3 NOTATION

Für die in dieser Arbeit vorkommenden mathematischen Ausdrücke wird die folgende Notation genutzt:

$a, b, c, A, B, C, ...$	Skalar
$\mathbf{a}, \mathbf{b}, \mathbf{c}, ...$	Spalten-Vektor
$\mathbf{A}, \mathbf{B}, \mathbf{C}, ...$	Matrix bzw. Tensor
$\mathbf{a}^T$	Transponierung des Vektors $\mathbf{a}$
a_i	das i-te Element des Vektors $\mathbf{a}$
a_{ij}	das Element der Matrix $\mathbf{A}$ in der i-ten Zeile sowie j-ten Spalte
$a^{(l)}, \mathbf{a}^{(l)}, \mathbf{A}^{(l)}$	das l-te Element einer Menge von Skalaren, Vektoren oder Matrizen
$\lvert\mathbf{a}\rvert$	euklidische Norm des Vektors $\mathbf{a}$

2 GRUNDLAGEN

Um den Forschungsfragen nachzugehen und sie zu beantworten, müssen die Bestandteile des Erkennungssystems und der Self-Training Methode dargestellt und erklärt werden. Das in dieser Arbeit verwendete Neuronale Netz **CoMER** (vgl. Abschnitt 3.1.1) ist ein Transformer-basiertes Enkoder-Dekoder Modell, welches den Coverage Mechanismus aus dem Gebiet der Rekurrenten Neuronalen Netze in die Transformerarchitektur integriert und mathematische Formeln bidirektional erkennt. Zur Merkmalsextraktion werden Faltungsschichten aus Faltungsnetzen verwendet und das Netzwerk wird mithilfe von Stochastic Gradient Descent angelernt. Die genannten Einzelteile werden nun im Folgenden näher behandelt und erklärt.

Neuronale Netze sind dabei nur ein Untergebiet der Mustererkennung. Das übergreifende Forschungsgebiet befasst sich nach Niemann „mit den mathematisch-technischen Aspekten der automatischen Verarbeitung und Auswertung von Mustern“ [Nie83, 13-14]. Dabei wird „für ein physikalisches Signal (z. B. Sprache, Bild, Messwert) eine geeignete Symbolkette (bzw. eine formale Datenstruktur) berechnet“ [Nie83, 13-14]. Intuitiv wird einer definierten Eingabe eine Kette von Ausgaben zugeordnet. Übertragen auf die mathematische Formelerkennung entspricht dies der Zuordnung eines Eingabebildes einer Formel zu einer Sequenz an Formelzeichen. Der allgemeine Aufbau eines Systems zur Mustererkennung kann in Abbildung 2.0.1 anhand der Beispielhaften Transkription einer einzelnen mathematischen Formel nachvollzogen werden. Die Aufnahme, Digitalisierung und Vorverarbeitung sind meist einmalige Schritte zur Erstellung eines Datensatzes.

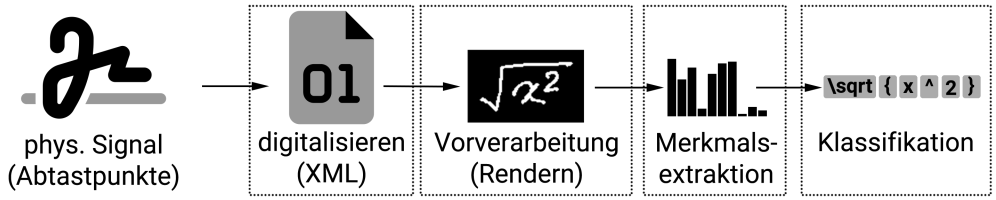


Abbildung 2.0.1: *Übersicht der Abläufe in einem Mustererkennungssystem, verdeutlicht mit der Transkription eines Bildes des CROHME$_{19}$ Datensatzes [MZM+19].*

Die CROHME [MVGZG14, MVGZG16, MZM+19] Datensätze bestehen z. B. aus handgeschriebenen mathematischen Formeln, die aus dem physikalischen Signal der Abtastpunkte der Schreibtrajektorie digitalisiert und im InkML [WU11] Format abgespeichert worden sind. Die Trajektorien werden für Online-Erkennungsaufgaben verwendet, in welchen die Eingabe der Echtzeit-Nutzereingabe entspricht. Ein Beispiel dafür ist die Nutzereingabe auf einem Tablet. Die Offline-Erkennung verwendet dahingegen Bilder, sodass beispielsweise archivierte Dokumente transkribiert werden sollen. Das InkML Format kann in einem Vorverarbeitungsschritt, wie in Abbildung 2.0.1 dargestellt, durch Rendern in ein Bild konvertiert werden (vgl. Abschnitt 5.1.1). Die Merkmalsextraktion sowie Klassifikation stellen eine Art Optimierungsproblem hinsichtlich der Approximation einer unbekannten Klassifikationsfunktion dar. Die beiden Schritte können durch separate Systeme gelöst werden oder gemeinsam in einem Ende-zu-Ende System.

2.1 NEURONALE NETZE

Neuronale Netze sind Beispiele für Ende-zu-Ende Systeme zur Merkmalsextraktion und Klassifikation innerhalb der Pipeline der Mustererkennung [LBBH98, CSG18]. Wie im Namen bereits enthalten, handelt es sich um ein Neuronen-Netz, also Neuronen, die entsprechend des Netzwerktyps miteinander verbunden sind. Das Ziel ist es eine beliebige Funktion zu approximieren [LBBH98, Hor91]. Die Zielfunktion kann beispielsweise die perfekte Klassifikation eines einzelnen Zeichens auf Grundlage des Eingabebildes sein. Die Ausgabe des Netzwerks ist dabei eine Verteilung über das verwendete Alphabet und soll ein Maximum beim korrekten Zeichen haben.

Ein Neuron bezeichnet den kleinsten Baustein eines Netzwerks und kann wie das biologische Pendant verstanden werden: Eingehende Verbindungen können zum Auslösen des Neurons führen, sodass mit dem Ausgang verbundene Neuronen gegebenenfalls auch auslösen. Falls die Neuronen des gesamten Netzwerks in Schichten sortiert werden können, sodass der Informationsfluss unidirektional ist und alle Neuronen einer Schicht mit allen direkt darauffolgenden Verbunden sind (vollverbunden), handelt es sich um ein *feedforward Neural Network* oder *Multi-Layer Perceptron* (**MLP**) [Hay98]. MLPs werden in reiner Form selten verwendet, jedoch werden sie meist am Ende eines Netzwerks zur Berechnung der Ausgabeverteilung eingesetzt [LKJ16]. Davor können problemspezifische Schichten verwendet werden, die oftmals auch weniger Speicher und Berechnungen benötigen. In der Domäne von visuellen Daten sind das unter anderem Faltungsschichten (vgl. Abschnitt 2.2). Wie in [Hor91] gezeigt, kann

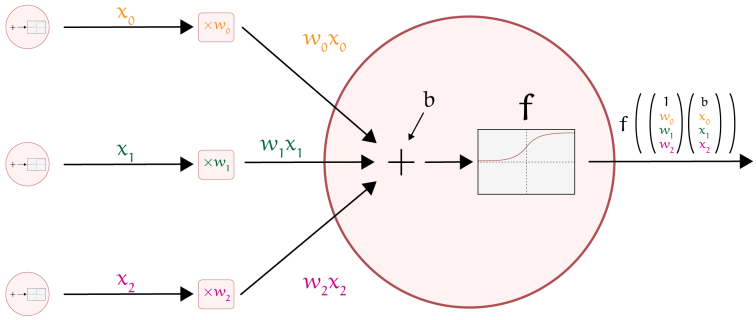


Abbildung 2.1.1: *Darstellung eines Neurons und der Berechnung der Ausgabe mithilfe einer nicht-linearen Funktion und einem Skalarprodukt.*

jedoch bereits alleinig ein MLP mit jeweils einer Eingabe-, Ausgabe- und versteckten Schicht eine beliebige Funktion approximieren, wenn beliebig viele Neuronen in der versteckten Schicht existieren dürfen und eine nicht-lineare Aktivierungsfunktion verwendet wird.

Wie in Abbildung 2.1.1 zu erkennen, werden eingehende Verbindungen gewichtet, aufsummiert und zu einem *Bias* addiert. Auf die Summe wird anschließend eine nicht-lineare Aktivierungsfunktion angewendet, damit die Berechnung nicht durch eine einzige Matrixmultiplikation ersetzt werden kann und das Netzwerk auch nicht-lineare Funktionen approximieren kann [GBC16, Hor91].
Sei $\mathbf{W}_1$ beispielsweise die Gewichtsmatrix einer Schicht Neuronen eines MLPs, welche alle mit der gleichen Eingabe $\mathbf{E}$ verbunden sind, jedoch potentiell andere Gewichte verwenden. Der Bias kann dabei in $\mathbf{W}$ durch das erste Element jeder Zeile (Gewichtsvektor eines einzelnen Neurons) und einer konstanten 1 als erstes Element im Eingabevektor realisiert werden (vgl. Abbildung 2.1.1). Ohne Aktivierungsfunktion kann die Ausgabe $\mathbf{A}_1$ und die Ausgabe einer darauffolgenden Schicht $\mathbf{A}_2$ wie folgt berechnet und mit einer einzigen Matrixmultiplikation ersetzt werden:

$$\begin{aligned} &\mathbf{W}_1 \cdot \mathbf{E} = \mathbf{A}_1,\ \mathbf{W}_2 \cdot \mathbf{A}_1 = \mathbf{A}_2,\ \mathbf{W}_2 \cdot \mathbf{W}_1 = \mathbf{W}_z \\ &\quad \Rightarrow \mathbf{W}_2 \cdot \mathbf{W}_1 \cdot \mathbf{E} = \mathbf{A}_2 \\ &\quad\quad \Rightarrow \mathbf{W}_z \cdot \mathbf{E} = \mathbf{A}_2 = \mathbf{W}_2 \cdot \mathbf{W}_1 \cdot \mathbf{E}\,. \end{aligned}$$

Durch die Einführung der nicht-Linearität kann diese Vereinfachung nicht durchgeführt werden und die Funktion, welche das Netzwerk oder MLP repräsentiert,

kann entsprechend auch nicht-linear sein [Hor91]. Neben der Sigmoid Funktion $\sigma(x) = (1 + e^{-x})^{-1}$ ist $\mathrm{ReLU}(x) = max(0, x)$ eine häufige Wahl, da bei ihrer Verwendung ein schnelleres und stabileres Anlernen möglich ist [KSH12].
Außerdem kann zur Ausgabe einer Pseudowahrscheinlichkeit eine Softmax-Schicht eingesetzt werden, welche die Aktivierungen der vorherigen Schicht in dem Intervall $[0, 1]$ begrenzt. Die Softmax-Funktion ist dabei wie folgt definiert:

$$\mathrm{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.1.1}$$

Zu erkennen ist, dass die Ausgabe, unabhängig von der Eingabe im Intervall $[0, 1]$ liegt. Ein einzelner Wert der Ausgabe kann als Konfidenz verstanden werden. Je nach Kalibrierung des Netzwerks ist der numerische Wert aber nicht unbedingt aussagekräftig (vgl. Abschnitt 3.3 und Ergebnisse in Abschnitt 5.5). Besonders ReLU-Netzwerke neigen zur *Overconfidence*, wodurch viele falsche Vorhersagen einen hohen Konfidenzwert erreichen [HAB19].

2.1.1 *Backpropagation & Stochastic Gradient Descent*

Die verwendeten Aktivierungsfunktionen müssen differenzierbar sein, damit die Optimierungsstrategien partielle Fehlergradienten verwenden können [GBC16]. Der Fehler wird dabei durch die *Loss-Funktion* ϵ berechnet und ist ein Abstandsmaß zwischen der Netzwerkausgabe und dem gewünschten Ergebnis. *Stochastic Gradient Descent* (SGD) nutzt dann die partiellen Fehlergradienten $\partial\epsilon/\partial w_{ij}^{(l)}$ eines einzelnen Neuron-Gewichts zur Minimierung des Fehlers durch inkrementelle Änderung der Gewichte. Wie in [Nie83, 388] beschrieben, kann die Update-Regel mit

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \beta \frac{\partial \epsilon}{\partial w_{ij}^{(l)}} \tag{2.1.2}$$

definiert werden. Durch β wird die „Schrittweite“ des Gradientenabstiegs vom Nutzer gesteuert. Üblicherweise wird β im Laufe des Trainings reduziert, um ein lokales bzw. globales Optimum immer genauer zu erreichen. Oftmals wird die Update-Regel für Mini-Batches durchgeführt, sodass der Gradient über mehrere Datenpunkte des Trainingsdatensatzes gemittelt wird [GBC16, 275].
Backpropagation bzw. *Error-Back-Propagation* wird dabei zur effizienten Berechnung von Fehlergradienten der Gewichte verwendet. Durch Ausnutzung der Kettenregel und Rückführung der Abhängigkeit des lokalen Fehlers $\delta_j^{(l)}$ einer Schicht auf den darauffolgenden ist eine schrittweise Berechnung möglich. Der lokale Fehler der

letzten Schicht $\delta_j^{(L)}$ wird wie folgt direkt berechnet, wonach die vorherigen Schichten schrittweise folgen [Nie83, 389]:

$$\begin{aligned}
\delta_j^{(L)} &= (d_j - f_j^{(L)})(1 - f_j^{(L)}) f_j^{(L)} \\
\delta_j^{(l)} &= \left\{ \sum_{k=0}^{M^{l+1}-1} w_{jk}^{(l+1)} \delta_k^{(l+1)} \right\} (1 - f_j^{(l)}) f_j^{(l)}, \forall 0 \leqslant l < L \\
\frac{\partial \epsilon}{\partial w_{ij}^{(l)}} &= \delta_j^{(l)} f_i^{(l-1)}
\end{aligned} \tag{2.1.3}$$

Dabei ist $f_j^{(l)}$ die Ausgabe eines Neuron j von Schicht l, M^l die Anzahl an Neuronen in einer Schicht und $\mathbf{d}$ die gewünschte Ausgabe des Neurons der letzten Schicht. Zu erkennen ist hier die Berechnungsvorschrift des Fehlergradienten $\partial\epsilon/\partial w_{ij}^{(l)}$ auf Grundlage des lokalen Fehlers, der entweder direkt oder in Abhängigkeit des nachfolgenden lokalen Fehlers berechnet werden kann.
Bei der Herleitung der Formeln 2.1.3 wurde der Mean-Squared-Error $\epsilon_{MSE} = \frac{1}{2} \cdot \sum_{i=0}^{M^L-1} (d_i - f_i^{(L)})^2$ als Loss-Funktion und eine Sigmoid Aktivierungsfunktion verwendet. Die Herleitung kann aber analog mit anderen Loss- bzw. Aktivierungsfunktionen durchgeführt werden. Je nach Problemdomäne und Präferenz kann die Loss-Funktion variieren [WMZT22]. Das in dieser Arbeit verwendete Neuronale Netz CoMER wird beispielsweise mit dem Cross-Entropy Loss

$$\epsilon_{CE} = - \sum_{i=0}^{M^L-1} (d_i \log(f_i^{(L)}) + (1 - d_i) \log(1 - f_i^{(L)}) , \tag{2.1.4}$$

ReLU und SGD angelernt.

Die Approximation der Zielfunktion (perfekte Klassifikation) wird aber nur implizit durch die Minimierung des Fehlers erreicht. Das gewünschte Ziel wird üblicherweise mit einer Evaluationsmetrik auf einem Testdatensatz gemessen. Dies kann beispielsweise die fehlerfreie Erkennungsrate einer gesamten Formel sein. Dadurch ergeben sich zwei Ziele, wobei nur die Minimierung des Fehlers direkt optimiert wird. In der Praxis korrelieren die beiden Ziele in gewissem Maße, sodass mit einem minimalen Loss auch eine gute Leistung hinsichtlich der Evaluationsmetrik einhergeht [GBC16, 273]. Darüber hinaus ist die „wahre“ Verteilung der Daten im Produktionsbetrieb i. d. R. unbekannt, wodurch die Minimierung des Fehlers auf den Trainingsdaten lediglich das empirische Risiko minimiert [GBC16, 272ff.]. Anders ausgedrückt sollten daher sowohl die Trainings- als auch die Testdaten möglichst gut die Realität widerspiegeln.

2.2 FALTUNGSNETZE

Neben den zuvor vorgestellten MLPs sind Faltungsnetze (*Convolutional Neural Networks*, CNNs) eine weitere Art Neuronen miteinander zu verbinden und zu nutzen. Die Ersatzfunktion f eines MLPs oder dessen versteckte Schichten hat eine Definitionsmenge und Bild von reellen Vektoren ($f : \mathbb{R}^a \to \mathbb{R}^b$). In Kontrast dazu können die in CNNs eingesetzten Faltungsschichten Tensoren als Ein- bzw. Ausgabe haben. Ein Tensor ist eine mehrdimensionale Matrix. Besonders in der Mustererkennung von visuellen Daten finden Faltungsschichten Anwendung [LKJ16]. Ein CNN beinhaltet neben den zuvor vorgestellten vollverbundenen Schichten die namensgebenden Faltungsschichten.
Ein Eingabebild im Format eines Tensors besteht aus drei Dimensionen (z. B. $\subseteq \mathbb{R}^{(h \times w \times 3)}$). Die ersten beiden entsprechen den räumlichen Dimensionen des regelmäßigen Pixelgitters des Bildes. Die letzte Dimension enthält schließlich die tatsächlichen Bilddaten. Üblicherweise werden Bilder im RGB-Format abgespeichert, sodass jeder Farbkanal in einer Ebene der dritten Dimension gespeichert wird. Ein Graustufenbild wiederum benötigt nur eine Ebene, wodurch die letzte Dimension nicht explizit aufgespannt werden muss. Faltungsschichten (*Convolutional Layer*) falten die Tensoreingabe mit einem anlernbaren Filter. Intuitiv soll der Filter visuelle Merkmale (wie z. B. eine vertikale Kante) durch die Faltung in der Eingabe erkennen und lokalisieren. Der Trainingsprozess sorgt für die Ausbildung von klassenspezifischen Merkmalen in den Filtern [LKJ16]. Die Idee stammt dabei aus der Signalverarbeitung und der Faltungsoperation „$*$“ [GBC16]. Die Faltung kann als Integral bzw. dessen Diskretisierung wie folgt definiert werden [Nie83, 89]:

$$\begin{aligned} h(x,y) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(u,v)g(x-u,y-v)\,du\,dv \\ h_{jk} &= \sum_{\mu=0}^{M_x-1}\sum_{\nu=0}^{M_y-1} f_{\mu\nu}\,g_{j-\mu,k-\nu} \\ h &= f * g \end{aligned} \tag{2.2.1}$$

Wie zu erkennen ist, bildet die diskrete Faltung die Summe des Hadamard Produkt [Mil07] an jeder Position der Eingabe, auf welche der Filter angewendet wird. Eine Faltungsschicht ist definiert durch die Filter bzw. *Kernelgröße* K, *Filteranzahl* F, *Padding* P und *Stride* S, und erzeugt einen Ausgabetensor mit den Dimensionen $W' \times H' \times F$, wobei

$$W' = \frac{W-K+2P}{S}+1 \text{ und } H' = \frac{H-K+2P}{S}+1 \tag{2.2.2}$$

gilt [LKJ16]. Entsprechend des Padding Parameters wird die Eingabe in den ersten zwei Dimensionen vergrößert, damit die Ausgabe beispielsweise eine bestimme Dimension erreicht. Neben dem Auffüllen von Nullen können auch andere Strategien, wie das Spiegeln des Tensors, verwendet werden [LKJ16]. Die Stride steuert die Schrittweite des Filters. Wie in Abbildung 2.2.1 zu sehen, wird der Filter bzw. Kernel im zweiten Berechnungsschritt entsprechend der Stride eine Position nach rechts verschoben. Eine größere Stride überspringt Elemente der Eingabe und verringert die Dimension der Ausgabe (vgl. Gleichung 2.2.2).

Zusätzlich ist in Abbildung 2.2.1 ein einfaches Beispiel zur diskreten Faltung dargestellt. Nach und nach bzw. hochgradig parallelisiert auf *Graphics Processing Units* (GPUs) wird so die Ausgabe der Schicht berechnet. Im Kontext von Neuronen und deren Gewichten kann jedes Ausgabeelement als Neuron angesehen werden, welches mit den verschiedenen Punkten der Eingabe verbunden ist. Im Kontrast zu MLPs sind Gewichte der Verbindungen aber geteilt. Jedes Ausgabeneuron teilt die Filtergewichte mit den Neuronen der gleichen Ebene. Dadurch benötigen Faltungsschichten tendenziell wenig anlernbare Parameter (Gewichte), auch wenn deutlich größere Eingaben verarbeitet werden. Die benötigte Anzahl an verwendeten Parametern lässt sich über $K \cdot K \cdot D \cdot F$ berechnen [LKJ16], wenn D die Größe der dritten Dimension der Eingabe beschreibt. Die Gewichte der Schicht können analog, wie zuvor beschrieben, mit SGD angelernt werden. Die Aktivierungsfunktion ist dabei nicht Teil der Schicht, weswegen üblicherweise eine Aktivierungsschicht hinter einer Faltungsschicht hinzugefügt wird, welche lediglich die Aktivierungsfunktion auf jeden Wert der Eingabe anwendet.
Ein weiterer Vorteil der Faltungsschicht ist die Invarianz des Filters bezüglich der Position. Da der Filter potentiell an jeder Stelle der Eingabe angewendet wird, können Merkmale der Eingabe unabhängig von ihrer Position hervorgehoben werden. Der Filter ist räumlich limitiert durch die Filtergröße. Mithilfe von *Pooling-Schichten* kann das effektiv betrachtete Umfeld vergrößert werden. Durch vorheriges Herunterskalieren der Eingabe wird ein größerer effektiver Bereich abgedeckt, wenn die Filtergröße unverändert bleibt. Eine Pooling-Schicht funktioniert dabei annähernd analog zu einer Faltungsschicht ohne anlernbaren Filter. Jedoch wird das Maximum oder der Durchschnitt im Bereich des Filters für jede Ebene der dritten Dimension einzeln gebildet. Eine übliche Architektur wechselt zwischen mehreren Faltungsschichten und einer Pooling-Schicht [LBBH98, ZG22a, KSZQ19]. Durch die Pooling-Schicht wird die Größe des Tensors verringert, sodass nachfolgende Schichten, wie z. B. ein MLP zur Klassifikation, weniger Neuronen bzw. anlernbare Parameter benötigen.

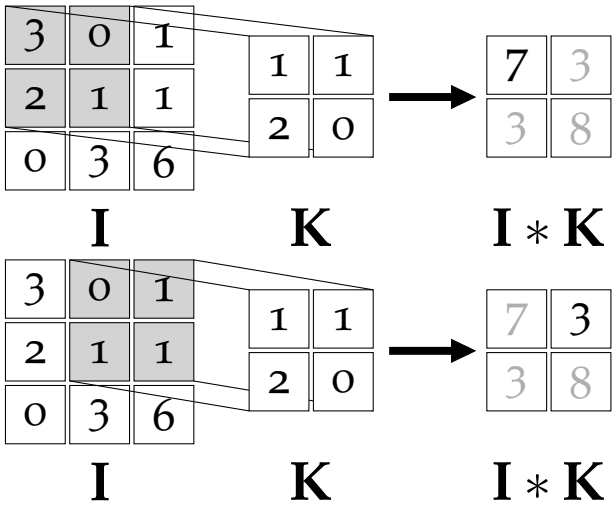


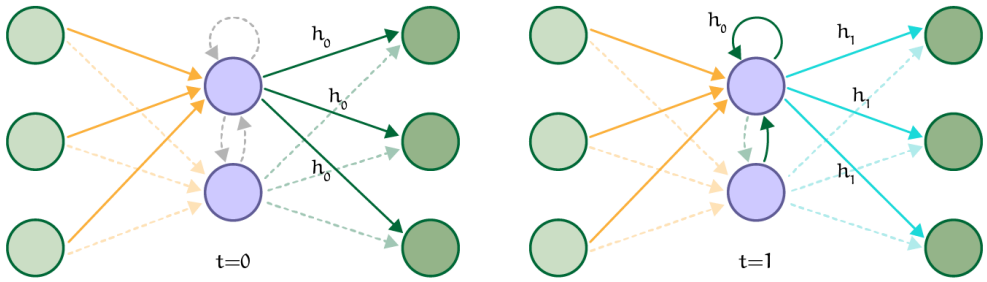
Abbildung 2.2.1: *Eine Faltungsschicht* ($K = 2, F = 1, P = 0, S = 1$)*, welche eine Eingabe* $\mathbf{I}(3 \times 3)$ *mit einem Filter* $\mathbf{K}(2 \times 2)$ *faltet. Das Hadamard Produkt wird entsprechend der Stride an verschiedenen Stellen gebildet und anschließend aufsummiert.*

2.3 REKURRENTE NEURONALE NETZE

Bisher handelt es sich in den vorgestellten Netzwerken um feedforward Networks (vgl. Abschnitt 2.1), in welchen der Informationsfluss nur unidirektional ist. Im Gegensatz dazu sind in Rekurrenten Neuronalen Netzwerken (RNNs) die namensgebenden rekurrenten Verbindungen zwischen Neuronen erlaubt [RHW86]. In Abbildung 2.3.1b sind die rekurrenten Verbindungen in grün eingezeichnet und die Aktivierungen in Richtung der Ausgabeschicht lassen sich mit

$$\mathbf{h}_t = \Theta(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1}) \tag{2.3.1}$$

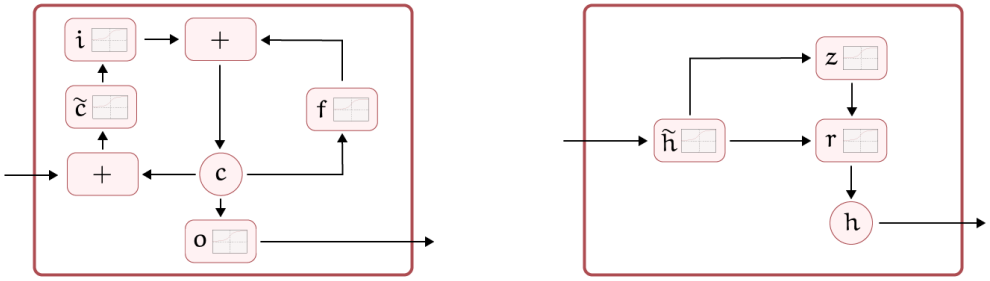
berechnen, wobei Θ die verwendete Aktivierungsfunktion ist [Gra12b]. Mit $\mathbf{U}$ sind die Gewichte der rekurrenten und mit $\mathbf{W}$ die Gewichte der restlichen Verbindungen gegeben. Zu erkennen ist die Abhängigkeit von der Aktivierung des letzten Zeitschritts $\mathbf{h}_{t-1}$ in der Berechnung des derzeitigen $\mathbf{h}_t$. So hängt die Berechnung von $\mathbf{h}_1$ in Abbildung 2.3.1b beispielsweise von der vorherigen Aktivierung $\mathbf{h}_0$ ab. Durch das hinzufügen der zeitlichen Komponente können RNNs analog zur universellen Approximationsleistung des MLPs sequenzbasierte Probleme beliebig genau lösen bzw. die Zuordnung der Sequenzen beliebig genau approximieren [Hamoo]. RNNs finden daher eine breite Anwendung in der Problemdomäne von sequenzierbaren Daten, wie z. B. Text, gesprochene Sprache oder in der Signalverarbeitung [RS21].



(a) (b)

Abbildung 2.3.1: *Beispiel zweier Zeitschritte eines RNN Netzwerks. Im ersten Schritt (a) sind die grauen rekurrenten Verbindung nicht aktiv, da keine vorherige Aktivierung vorliegt. Die Aktivierung h_0 wird zur Ausgabeschicht weitergereicht und wird im zweiten Zeitschritt (b) für die Berechnung von h_1 verwendet.*

Die zuvor genutzte Backpropagation zur effizienten Bestimmung von Fehlergradienten ist jedoch nicht direkt auf ein RNN übertragbar. *Backpropagation-Through-Time* (BPTT) nutzt *Unrolling* [RHW86], um ein äquivalentes feedforward Network auf Grundlage eines RNNs und einer Zykluslänge zu erhalten, und wendet darauf den Backpropagation Ansatz an [WZ95]. Dadurch wird mit dem letzten Zeitschritt und der Ausgabeschicht begonnen und von dort an rückwärts propagiert. Ein Gewicht des RNNs tritt dabei in dem entrollten Netzwerk häufiger auf und die Gradienten werden aufsummiert. Es existieren weitere Ansätze, welche die Zwischenzustände beim Entrollen nicht im Speicher halten müssen. Einer davon ist z. B. *Real-Time Recurrent Learning*, welcher zwar eine feste Menge Speicher (kubisch in der Zustandsgröße) benötigt, aber auch quartisch viele Berechnungen in Abhängigkeit von der Parameterzahl [WZ89]. In der Praxis haben sich unter anderem lange Abhängigkeiten zwischen Teilen der Eingabesequenz und schwierige Gradienten (verschwindende oder explodierende) beim Anlernen als Problem dargestellt [HS97, BSF94]. *Long Short-Term Memory* (LSTM) [HS97] oder die *Gated Recurrent Unit* (GRU) [CMBB14] versuchen diese Probleme zu lösen. Konzeptionell wird ein Neuron mit einer steuerbaren *Memory-Cell* ersetzt. Zuvor wird die Aktivierung der rekurrenten Verbindung immer vollständig in jedem Zeitschritt ersetzt (vgl. Abbildung 2.3.1).
Im Falle einer LSTM Zelle „steuern“ nun drei Faktoren (*Gates*) den inneren Zustand: Wie viel des Speichers ausgegeben, vergessen oder mit neuem Inhalt überschrieben wird. Die Berechnung der einzelnen Faktoren wird, wie in Formel 2.3.1, unter Hinzunahme des gewichteten inneren Zustands berechnet, sodass sich z. B. der Ausgabefaktor zu $\mathbf{o}_t = \Theta(\mathbf{W}_o\mathbf{x}_t + \mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{V}_o\mathbf{c}_t)$ ergibt [CGCB14]. Die Faktoren werden



(a) *Long-Short Term Memory Zelle* **(b)** *Gated Recurrent Unit*

Abbildung 2.3.2: *Aufbau einer (a) LSTM Zelle bzw. (b) GRU in Anlehnung an [CGCB14]. Mithilfe mehrerer unstetiger Funktionen und anlernbaren Gewichtsparametern wird der innere Zustand* $\mathbf{c}$ *bzw.* $\mathbf{h}$ *in einem Berechnungsschritt aktualisiert.*

also mithilfe von anlernbaren Gewichtsmatrizen sowie nicht-linearen Aktivierungsfunktionen berechnet. Die Ausgabe bzw. Aktivierung der LSTM Zelle entspricht dann $\mathbf{h}_t = \mathbf{o}_t \Theta(\mathbf{c}_t)$ [CGCB14].
Die GRU hat einen einfacheren Aufbau als die LSTM Zelle und verwendet nur zwei steuerbare Gates. Die GRU gibt immer den vollständigen inneren Zustand aus und besitzt ein Update- sowie Reset-Gate. Der Aufbau beider Zellen ist in Abbildung 2.3.2 dargestellt und zeigt die verschiedenen Gates und wie diese den inneren Zustand der Zelle beeinflussen. RNNs mit LSTMs wurden und werden derzeitig im Bereich der Sequenzvorhersage, des Sequenz-Labeling (Kategorisierung von Teilen einer Sequenz), des Natural Language Processing (NLP) und vielen weiteren verwendet [RS21].

2.3.1 *Enkoder-Dekoder*

Einer der weiteren Bereiche ist die Anwendung von RNNs in *Enkoder-Dekoder* Netzwerken [OMK21]. Das zu lösende Problem ähnelt dem Enkodierungs-Problem, welches die Abbildung einer Eingabe auf eine Vektorrepräsentation und zurück auf die identische Eingabe beschreibt [RHW86]. Eine Architektur zum Lösen des Problems ist der *Autoenkoder* [Kra91], bestehend aus einem Enkoder und Dekoder. Zwischen diesen befindet sich üblicherweise ein „Flaschenhals“, wodurch die eingegebenen Informationen im Enkoder reduziert und kodiert werden müssen, um diese im Dekoder zur Generierung der Ausgabe zu verwenden. Im Kontrast dazu werden Enkoder-Dekoder Netzwerke zum Lösen von *X-to-Y* Problemen verwendet, in welchen die gewünschte Ausgabe nicht identisch mit der Eingabe ist [SVL14]. Ansonsten entspricht der grundlegende Aufbau der Enkoder-Dekoder Architektur der eines Autoenkoders.

Besonders im Bereich der Sprachübersetzung werden Sequence-to-Sequence Enkoder-Dekoder Modelle verwendet, um eine Wortsequenz einer Sprache auf eine Wortsequenz in einer anderen Sprache abzubilden [OMK21]. Das erste Enkoder-Dekoder Modell von Google [SVL14] nutzt dabei sowohl für den Enkoder als auch Dekoder ein RNN mit LSTM Zellen. Abseits der Sprachübersetzung werden Enkoder-Dekoder Netzwerke besonders im Bereich des Natural Language Processing, aber auch in der Natural Language Generation angewendet [OMK21]. Letzteres beinhaltet z. B. die Generierung von Bildbeschreibungen, welches ein Image-To-Sequence Problem darstellt. Zur Image-To-Sequence Problemdomäne zählt auch die Transkription von handgeschriebenen Texten bzw. mathematischen Formeln. Viele derzeitige Ansätze in diesem Bereich nutzen dabei eine veränderte Enkoder-Dekoder Architektur (vgl. Abschnitt 3.1).

2.3.2 *Attention & Coverage*

Zur weiteren Verbesserung der Übersetzungsgenauigkeit der Enkoder-Dekoder Architektur wurde z. B. in [BCB15] oder [LPM15] ein *Attention* Modell hinzugefügt. Ein Kontext-Vektor $\mathbf{c}_t$ soll dabei relevante Informationen für einen einzelnen Dekodierungsschritt in Abhängigkeit von der (kodierten) Eingabe mit

$$\mathbf{c}_t = \sum_i \alpha_{ti} \mathbf{h}_i \tag{2.3.2}$$

berechnen [BCB15]. In dem Ansatz werden die versteckten Zustände der kodierten Eingabe $\mathbf{h}_s$ zur Berechnung der Attention-Gewichte α_{it} genutzt. Zuvor wurden diese in [SVL14] nur zum Initialisieren des Zustands im Dekoder verwendet [LPM15]. Die Berechnung der Gewichte kann in globale und lokale Ansätze eingeteilt werden, wodurch der beeinflussende Bereich begrenzt oder unbegrenzt ist. Ein globaler Ansatz aus [ZDD18] verwendet beispielsweise

$$\begin{aligned} (\mathbf{e}_t)_i &= \mathbf{v}_\alpha^T \tanh(\mathbf{W}_\alpha \mathbf{h}_t + \mathbf{U}_\alpha \mathbf{a}_i) \\ \alpha_{ti} &= \frac{\exp(e_{ti})}{\sum_k^L \exp(e_{tk})} \end{aligned} \tag{2.3.3}$$

zur Berechnung der Attention-Gewichte auf Grundlage der Eingabe $\mathbf{U}_\alpha \mathbf{a}_i$. Die Eingabe ist mit den versteckten Zuständen $\mathbf{h}_s$ vergleichbar, nur handelt es sich bei [ZDD18] um Merkmalsvektoren von Bilddaten statt den versteckten Zuständen einer Sequenz. Wie in [ZDZ+17, ZDD18] gezeigt, hat die Verwendung des Attention-Mechanismus auch in der Problemdomäne von Bilddaten eine Verbesserung erzielt.

Zusätzlich zur Attention kann, wie in [ZDZ+17], ein *Coverage*-Term [TLL+16] zu der Berechnung der Attention-Gewichte hinzugefügt werden. Das Ziel dabei ist die Verbesserung der vollständigen Abdeckung der Eingabe [TLL+16]. Kein Bereich der (kodierten) Eingabe soll während der Sequenzvorhersage ignoriert oder mehrfach abgedeckt werden. Der Coverage-Term ergibt sich durch Aufsummieren aller Attention-Vektoren vorheriger Zeitschritte gefaltet mit einer Gewichtsmatrix ($\mathbf{F} = \mathbf{Q} * \sum_{l=1}^{t-1} \mathbf{a}_l$). Die Einträge der Matrix werden dann in der Berechnung der neuen Attention-Gewichte verwendet. Dadurch erweitert sich z. B. Formel 2.3.3 nach [ZDD18] zu

$$(\mathbf{e}_t)_i = \mathbf{v}_\alpha^T \tanh(\mathbf{W}_\alpha \mathbf{h}_t + \mathbf{U}_\alpha \mathbf{a}_i + \mathbf{U}_f \mathbf{f}_i) \; . \tag{2.3.4}$$

In [ZDZ+17] wird alleinig durch das Hinzufügen des Coverage-Terms eine signifikante Verbesserung erreicht, wodurch in [ZDZ+17] zum Veröffentlichungszeitpunkt State-of-the-Art Ergebnisse im Bereich der handschriftlichen mathematischen Formelerkennung erzielt werden.

2.4 TRANSFORMER

Die zuvor behandelten RNNs und die Methoden zur Erweiterung arbeiten sequentiell. Schrittweise wird die kodierte Eingabe sowie die dekodierte Ausgabe erzeugt. Im Kontrast dazu lösen *Transformer* [VSP+17] das Sequence-to-Sequence Problem parallel. Ursprünglich wurde die Transformer-Architektur im Bereich der Sprachverarbeitung bzw.- übersetzung verwendet, jedoch ist diese auch auf andere Problemdomänen übertragbar (vgl. Abschnitt 3.1). Dadurch kann in einem einzelnen Schritt die vollständige Zielsequenz angelernt werden. Zusätzlich können durch kürzere Pfade im Attention-Mechanismus weitreichende Beziehungen zwischen den Eingaben besser angelernt werden [VSP+17]. Dafür nutzt die Architektur ausschließlich vollverbundene Schichten und die gleichzeitig eingeführte *Multi-Head Attention* (MHA). Insbesondere werden keine rekurrente Verbindungen oder Faltungsschichten verwendet.
Der Aufbau eines Transformers ist in Abbildung 2.4.1 dargestellt und zeigt den typischen Aufbau des Enkoder-Dekoder Modells mit einigen Veränderungen. Der Enkoder und Dekoder besteht nun aus mehreren Blöcken, wobei das *Positional Encoding* sowie der *Output*-Eingang zusätzliche Bestandteile sind. Durch den Wegfall des zyklischen Iterieren bis zu einem Abbruchzeichen fehlt ein variabler innerer Zustand und die davon abhängige Attention sowie Coverage. Zuvor konnte die Position während des En- bzw. Dekodieren implizit durch den inneren Zustand des RNNs repräsentiert werden. In der parallelisierten Architektur wird die Position in kodierter Form explizit zu

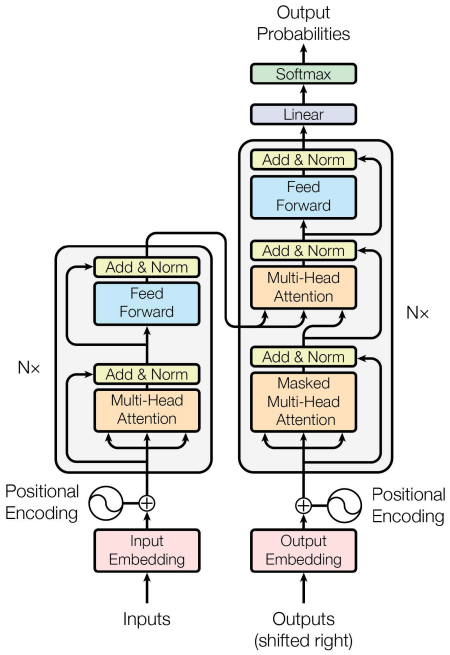


Abbildung 2.4.1: *Aufbau der Transformer-Architektur aus [VSP+17] mit einem Stapel an Enkoder-Blöcken auf der linken und einem weiteren Stapel an Dekoder-Blöcken auf der rechten Seite.*

der Enkodierung hinzugefügt. Nach [VSP+17] kann die Kodierung entweder statisch berechenbar oder angelernt sein, wobei die Autoren

$$\begin{aligned} PE_{(x,2i)} &= \sin(x \cdot 10000^{-2i/d_{\text{model}}}) \\ PE_{(x,2i+1)} &= \cos(x \cdot 10000^{-2i/d_{\text{model}}}) \end{aligned} \tag{2.4.1}$$

verwenden. Die Positions-Kodierung ist ein Vektor gleicher Dimensionalität wie die eingebetteten Eingabe- bzw. Ausgabe-Vektoren. Durch Aufsummieren ergibt sich ein neuer Vektor, der die Positions-Kodierung beinhaltet [VSP+17].
Der Output-Eingang (vgl. Abbildung 2.4.1) dient dazu, die *Teacher-Forcing* [WZ89] Methode anzuwenden und die Inferenz des Netzwerks durchzuführen. Die Annotation wird mit einem *Start-of-Sequence* (SOS) und *End-of-Sequence* (EOS) Zeichen erweitert. Teacher-Forcing nutzt dann die vollständige Annotation, mit Ausnahme des letzten Zeichens. Dieses soll vom Netzwerk vorhergesagt werden. Im Training soll das Netzwerk also bei „perfekten“ vorherigen Eingaben das gewünschte nächste Zeichen (z. B. EOS) vorhersagen. Ein zusätzliches Maskieren erlaubt ein paralleles Anlernen aller Teilsequenzen der Teacher-Forcing Eingabe. In der *Masked-Multi-Head-Attention* wird

dafür eine Dreiecksmaske verwendet, um sicherzustellen, dass bei der Vorhersage eines Zeichens keine Informationen von einer folgenden Position genutzt werden [VSP+17]. Dadurch kann in einem Schritt die Vorhersage jedes einzelnen Zeichens bei perfekten vorherigen Eingaben angelernt werden.

2.4.1 *Multi-Head Attention*

Der Kern der gesamten Architektur ist die Multi-Head Attention. Die Eingabe besteht aus den *Query*-, *Key*- und *Value*-Matrizen [VSP+17]. Im Kontext von Sprachübersetzungen erlaubt es jedem Eingabewort bzw. dessen Einbettung der Query, ein Attention-Gewicht zu jedem anderen Eingabewort der Key-Value Paare zu bilden. Intuitiv werden dadurch Wortbeziehungen möglich und trainiert, sodass bei der Übersetzung eines einzelnen Wortes der relevante Kontext des Wortes stark mit einbezogen werden kann. Die Eingabe wird in mehrere *Heads* aufgeteilt und zuvor mit

$$\text{head}_i(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \tag{2.4.2}$$

linear projiziert [VSP+17]. Das Aufteilen und Multiplizieren mit einer anlernbaren Gewichtsmatrix hat dabei bessere Ergebnisse erzielt als eine direkte Berechnung der Attention. Die eingesetzte *Scaled Dot-Product Attention* ist durch

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}})\mathbf{V} \tag{2.4.3}$$

definiert [VSP+17], wobei d_k der Dimensionalität der Key-Matrix $\mathbf{K}$ entspricht. Die Ergebnisse der einzelnen Berechnungen werden zu einer Matrix konkateniert und

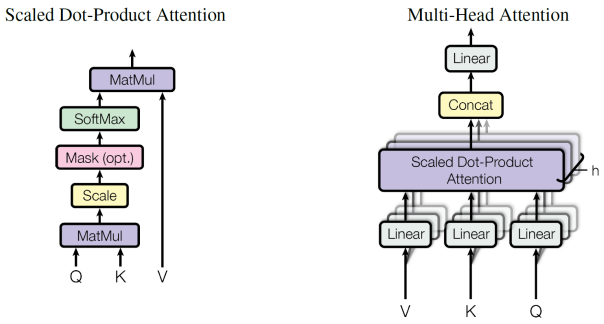


Abbildung 2.4.2: *Aufbau der Scaled Dot-Product Attention und Multi-Head Attention aus [VSP+17].*

anschließend erneut mithilfe einer vollverbundenen Schicht linear projiziert. Das beschriebene Vorgehen ist nochmals in Abbildung 2.4.2 abgebildet. Das Vorgehen ist analog auf visuelle Daten übertragbar, wobei das Enkodieren der Eingabe keine Wörter einbettet, sondern z. B. Bildausschnitte. Durch die Vorteile der Transformerarchitektur basieren derzeitig viele State-of-the-Art Modelle im Bereich des Natural Language Processing oder Enkoder-Dekoder Problemdomäne auf ihnen [HTB20, OMK21].

2.5 INFERENZ MIT BEAM-SEARCH

Die vorgestellten Netzwerke können in jedem Schritt eine bzw. mehrere Verteilungen über das gesamte Alphabet erzeugen. Die Verteilung stellt die Pseudowahrscheinlichkeit des nächsten Zeichens der Sequenz dar. Um aus der Verteilung nun eine vollständige Sequenz zu generieren, wird dieser Inferenzschritt in RNNs sowie Transformer Netzwerken zyklisch durchgeführt, bis eine Abbruchbedingung erfüllt ist. Das Netzwerk kann dabei als sequentieller Sequenzerkenner verstanden werden: Schrittweise wird jedes Zeichen einzeln erkannt und zur Ausgabesequenz hinzugefügt. Durch den Attention- bzw. Coverage-Mechanismus soll die Lokalisierung des nächsten Zeichens in jedem Schritt verbessert werden.

Da Transformer keinen inneren Zustand wie RNNs haben, wird die derzeitige Teilsequenz im Output-Eingang eingegeben (vgl. Abbildung 2.4.1). Durch die Erweiterung der Annotation um das SOS und EOS Zeichen kann ein einzelner Schritt simplifiziert als Abbildung $(\text{Bild}, (\text{SOS}, x_1, \ldots, x_n)) \rightarrow (y_1, \ldots, y_{n+1})$ dargestellt werden. Wie zu erkennen, ist die Länge der Teacher-Forcing Eingabe $(\text{SOS}, x_1, \ldots, x_n)$ und der Ausgabe-Sequenz $(y_1, \ldots, y_{n+1})$ gleich. Wenn für das letzte Zeichen der Ausgabe $y_{n+1} = \text{EOS}$ gilt, wird die Inferenz beendet und die Sequenz $(y_1, \ldots, y_{n+1})$ entspricht der Ausgabe des Netzwerks. Jedoch ist die Ausgabe in einem Inferenzschritt keine Sequenz von Zeichen, sondern eine Matrix mit mehreren Verteilungen über das gesamte Alphabet für jedes Zeichen y_i. Durch die Maskierung in der Multi-Head Attention muss nur die Verteilung über y_{n+1} betrachtet werden, da die vorherigen Verteilungen konstant bleiben. Die Inferenz beginnt mit $(\text{Bild}, (\text{SOS})) \rightarrow (y_1)$ und nutzt die Verteilung über y_1, um die Folgesequenz (SOS, y_1) zu bilden. Dieser Vorgang wiederholt sich so lange, bis eine Abbruchbedingung (z. B. EOS Zeichen) erfüllt ist.

Ein naiver Ansatz könnte alle möglichen Folgesequenzen erweitern und somit den vollständigen Suchraum abdecken. Dadurch würde zwar immer die „optimale“ Sequenz (hinsichtlich der Konfidenz) gefunden werden, jedoch ist der benötigte

Rechenaufwand exponentiell. Die *Greedy-Search* wählt dahingegen in jedem Inferenzschritt das Zeichen mit der höchsten Pseudowahrscheinlichkeit, wodurch maximal eine Sequenz pro Inferenzschritt erweitert wird. Dies hat jedoch einen Nachteil bei unsicheren Klassifikationen, da ähnlich wahrscheinliche Sequenzen (hinsichtlich der Konfidenz) verworfen und somit potentiell „optimale“ Sequenzen nicht gefunden werden. Dieses Problem wird mit der *Beam-Search* [Gra12a] verringert. Die Grundidee erweitert in einem Inferenzschritt eine Menge der Größe B von aktiven Hypothesen und wählt aus allen möglichen Erweiterungen die besten B (Beam-Breite) als neue aktive Menge. Dadurch wird ein größerer Bereich im Suchraum abgedeckt, ohne die exponentiellen Berechnungskosten der naiven Suche tragen zu müssen. Zur Bewertung von Sequenzen kann z. B. die Summe der log-Pseudowahrscheinlichkeiten der einzelnen Zeichen mit einer Längennormalisierung genutzt werden [SVL14]. Die Suche wird beendet, falls keine aktiven Hypothesen existieren oder nicht mehr besser als die schlechteste vollständige Hypothese werden können [Gra12a]. Schließlich wird die Hypothese mit der höchsten Bewertung als Ausgabe des Netzwerks gewählt.

2.5.1 *Pruning-Methoden*

Da maximal B aktive Hypothesen zu einem Zeitpunkt existieren dürfen, können auch bei sehr eindeutigen Pseudowahrscheinlichkeiten bis zu $B-1$ „schlechte“ Hypothesen erweitert werden. Dadurch wird die Berechnungszeit in vielen Fällen verlängert [FAO17]. Zur Reduktion der Berechnungszeit werden daher nicht nur die besten B ausgewählt, sondern auch kleinere Mengen zugelassen, falls die restlichen Hypothesen deutlich unwahrscheinlicher sind. Dafür können verschiedene Strategien verwendet werden. In [FAO17] werden z. B. vier Entscheidungsregeln zum Verringern der aktiven Hypothesen verwendet.

Relative Threshold Pruning (Formel 2.5.1) verwirft alle Hypothesen, die um einen Faktor schlechter sind als die beste Hypothese. *Relative Local Threshold Pruning* (Formel 2.5.2) nutzt das gleiche Vorgehen, jedoch wird nur das letzte Zeichen statt der gesamten Sequenz bewertet. *Absolute Threshold Pruning* (Formel 2.5.3) nutzt statt eines Faktors einen konstanten Abstand. *Maximum Candidates* limitiert die maximale Anzahl an Erweiterungen einer einzelnen Teilsequenz, sodass mehr Vielfalt in der Beam-Search erzwungen wird. Die Entscheidungsregeln zum Verwerfen einer Hypothese ergeben sich nach [FAO17] zu:

$$\text{score}(\text{cand}) \leqslant \lambda_{\text{rel}} \max_{c \in C} \{\text{score}(c)\} \tag{2.5.1}$$

$$\text{score}_w(\text{cand}) \leqslant \lambda_{\text{local}} \max_{c \in C}\{\text{score}_w(c)\} \tag{2.5.2}$$

$$\text{score}(\text{cand}) \leqslant \max_{c \in C}\{\text{score}(c)\} - \lambda_{\text{abs}} \tag{2.5.3}$$

Dabei bezeichnet score(cand) die Bewertung einer gesamten und $\text{score}_w(\text{cand})$ die Bewertung des letzten Zeichens einer Hypothese. Die Menge aller möglichen Folgesequenzen ist durch C gegeben. Die Berechnungszeit der Inferenz mit einer Beam-Breite von 14 im Bereich der Sprachübersetzung konnte in [FAO17] dadurch um 43% verbessert werden, ohne dass sich die Leistung deutlich verschlechtert.
Diese Arbeit fügt das *Constant Pruning* hinzu, welches einen konstanten Schwellenwert zum Verwerfen nutzt. Die dazugehörige Entscheidungsregel zum Verwerfen von Hypothesen ist mit

$$\text{score}(\text{cand}) \leqslant \lambda_{\text{const}} \tag{2.5.4}$$

gegeben. Durch diese Entscheidungsregel kann die Berechnungszeit der Inferenz, besonders zu Beginn des Trainings, deutlich verbessert werden. Zum Ende des Trainings bzw. bei einem konvergierten Netz sind jedoch keine Verbesserungen durch die Anwendung der Regel messbar (vgl. Abschnitt 5.3.1).

3 VERWANDTE ARBEITEN

Für die verwendeten Methoden im Trainingsprozess sowie für die Netzwerkarchitektur sind mehrere verwandte Arbeiten die Grundlage. Dazu zählt der Aufbau des Neuronalen Netzes, die Methodik zum semi-überwachten Anlernen des Netzwerks und die Kalibrierung der Konfidenzbewertung. Zusätzlich gibt es in den verschiedenen Bereichen auch alternative Ansätze, mit denen die Ergebnisse dieser Arbeit verglichen werden können (vgl. Kapitel 5).

3.1 FORMELERKENNUNG MIT NEURONALEN NETZEN

Die Ansätze zum Lösen des Image-to-Sequence Problems (vgl. Abschnitt 2.3.1) für handschriftliche mathematische Formeln sind vielseitig und können in Online und Offline Erkennungssysteme eingeteilt werden. Online-Ansätze nutzen die Schreibtrajektorie, wohingegen Offline-Ansätze nur ein Bild als Eingabe verwenden. In der Online-Erkennung sind Zeitinformationen in der Schreibtrajektorie enthalten. Dadurch ist die Online-Erkennung ein einfacheres Problem als die Offline-Erkennung, da einzelne Linien bereits ein Zeichen ergeben oder mit anderen Linien in der zeitlichen Umgebung in Relation gesetzt werden müssen. Die Segmentierung der Buchstaben ist dadurch deutlich einfacher als bei der Transkription eines Bildes.
Im Allgemeinen kann der Erkennungsprozess in zwei Stufen eingeteilt werden: Die Erkennung von Symbolen und eine anschließende strukturelle Analyse [CY00b]. In der strukturellen Analyse werden die erkannten Zeichen in Beziehung gesetzt, um die korrekte Reihenfolge und Position (z. B. Hoch- bzw. Tiefstellung) vorherzusagen.

Online Erkennungssysteme wie [ML14, CY00a, KW07, ASB14, KRLP99] nutzen dabei eine vordefinierte Grammatik zur strukturellen Analyse. Die Modelle in [ML14, CY00a] benötigen jedoch vor der Erkennung der Symbole eine separate Segmentierung bzw. Gruppierung von mehreren geschriebenen Linien. Dafür werden die Abtastpunkte in Linien eingeteilt und anschließend mithilfe von z. B. Elastic Matching [CY98] erkannt. Dabei wird eine Teilmenge der Linien mit vordefinierten Vorlagen verglichen. Die Vorlagen werden zusätzlich deformiert, um variable Schreibstile zu repräsentieren. In [ML14] wird dafür eine veränderte Variante des Elastic Matching [ML10]

verwendet, welche Distanzwerte zu den Vorlagen ausgibt. Die minimale Distanz kann dann als Vorhersage des einzelnen Symbols verwendet werden. In [ZBC02, LP98] wird die Symbolerkennung als gegeben angesehen und es werden Baumtransformationen (Grammatiken auf Graphen) zur Generierungen einer validen LaTeX Sequenz durchgeführt. Dabei wird mit einem flachen Ausdruck-Baum begonnen, welcher alle Symbole mit ihrer Position enthält. Anschließend wird dieser in einem *Layout Pass* in verschiedene Regionen (z. B. Tief-, Hochstellung oder Brüche) eingeteilt und durch eine Baumtransformation in einen neuen Baum überführt. Weitere Transformationen bereinigen Fehler und erzeugen schließlich einen Baum, der direkt in eine LaTeX Sequenz umgewandelt werden kann. Die validen Baumtransformationen müssen dabei nicht manuell modelliert werden, sondern können, wie z. B. in [WYZ+21], durch ein Neurales Netzwerk erlernt werden.

In [KRLP99] werden sowohl für die Segmentierung als auch für die Symbolerkennung mehrere Hidden-Markov-Modelle (HMM) verwendet. Ein HMM ist dabei ein endlicher Automat mit verschiedenen Übergangswahrscheinlichkeiten im Inneren. Ein HMM erzeugt Beobachtungen in Abhängigkeit von den nicht beobachtbaren inneren Zuständen. Der Viterbi [Vit67] Algorithmus kann dann für jedes HMM genutzt werden, um die wahrscheinlichste Sequenz von Zuständen zu finden, welche das zu erkennende Zeichen produzieren würden. Durch den Start- bzw. End-Zustand werden die Zeichen segmentiert und die Produktionswahrscheinlichkeit des HMMs wird zur Klassifikation des Zeichens verwendet. Anschließend wird erneut eine Grammatik mit Baumtransformationen angewandt, um aus den erkannten und segmentierten Zeichen die Ausgabe zu erzeugen [KRLP99].

Für die Symbolerkennung kann auch eine Support-Vector-Machine (SVM), wie z. B. in [KW07], verwendet werden. Dabei soll die SVM die Merkmalsvektoren der Online-Eingabe mithilfe einer Kernel-Funktion in einem transformierten Raum linear voneinander trennen. Der transformierte Raum wird dabei implizit durch die Kernel-Funktion aufgespannt. Intuitiv wird in diesem Raum eine Hyperebene zwischen zwei Clustern von Daten bestimmt, die den maximalen gerichteten Abstand zu den beiden Clustern besitzt. Durch die Richtung im Abstandsmaß kann die Seite des Datenpunktes bezüglich der Hyperebene und damit die Klassifikation bestimmt werden. Falls die Cluster nicht linear separierbar sind, kann der Fehler, also der gerichtete Abstand von falsch klassifizierten Punkten, minimiert werden. Mehrere SVMs können dabei zur Klassifikation von mehr als zwei Klassen verwendet werden [Nie83].

Wie in [ASB14], können SVMs auch zur Layout-Analyse verwendet werden. Die durch HMMs erkannten Zeichen bilden Regionen, welche mithilfe mehrerer SVMs in horizontal, vertikal, hoch- bzw. tiefgestellt und enthaltend eingeteilt werden. Ein Bruch soll dabei z. B. in „vertikal“ und ein Wurzelzeichen in „enthaltend“ klassifiziert werden. Abermals wird anschließend eine Grammatik zur Erzeugung der Sequenz verwendet.

Im Bereich der Offline-Erkennung nutzen Ansätze wie [Le20, ZDZ+17, ZDD18, YLD+22, BQX+22, ZDY+20] Neuronale Netze in einer Enkoder-Dekoder Architektur für einzelne Schritte oder für das gesamte Erkennungssystem. In den eingesetzten RNNs hat sich der Attention und Coverage Mechanismus als wichtiger Bestandteil erwiesen. Auch wenn die Neuralen Netzwerke die vorherigen Ansätze deutlich übertreffen, finden Teile davon, wie z. B. Baumtransformationen in Baum-Dekodern aus [ZDY+20], Anwendung.

Der RNN basierte Ansatz wurde erstmals durch einen Transformer in [KRR+20] ersetzt. Im BTTR Modell [ZGY+21] wurde darauf aufbauend das bi-direktionale Anlernen hinzugefügt. Beim bi-direktionalen Anlernen wird die gewünschte Zielsequenz in beiden Schreibrichtungen für die Teacher-Forcing Eingabe verwendet. Das Netzwerk soll dadurch die Sequenzen von links-nach-rechts sowie rechts-nach-links erkennen können. Während der Inferenz kann die Sequenz (beginnend mit einem SOS oder EOS Zeichen) schrittweise durchgeführt werden (vgl. Abschnitt 2.5). Die Abbruchbedingung der Beam-Search ist dabei durch das Gegenstück zum Start-Zeichen gegeben. Beide Arbeiten haben damit jeweils State-of-the-Art Erkennungsraten zu der Zeit der Veröffentlichung erreicht.

Es existieren jedoch deutlich mehr Ansätze, welche in Übersichtsarbeiten wie z. B. [CYoob, HTB20] vorgestellt und miteinander verglichen werden. Im Allgemeinen werden viele Mustererkennungsmethoden aus dem Bereich der Sequenzerkennung und *Handwritten Text Recognition* in ein Erkennungssystem für handschriftliche mathematische Formeln integriert und ihre Erkennungsleistung untersucht.

3.1.1 *CoMER: Transformer mit Coverage*

Ein weiteres Offline-Erkennungssystem für die Erkennung von handschriftlichen mathematischen Formeln ist das CoMER Modell. Als Nachfolger zum erwähnten BTTR Modell [ZGY+21], erweitert CoMER [ZG22a] (**Co**verage information in the

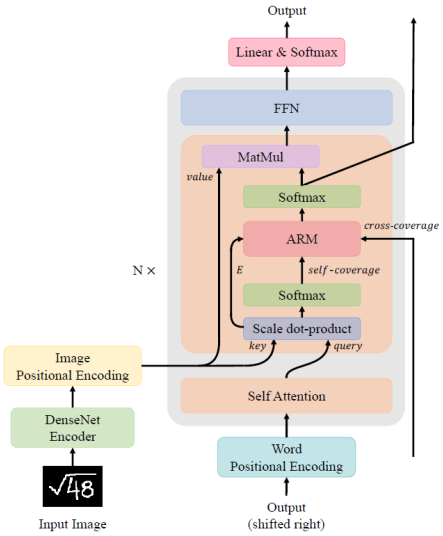


Abbildung 3.1.1: *Aufbau des CoMER Modells aus [ZG22a]. Die Multi-Head Attention des Transformers [VSP+17] wurde um das ARM Modul erweitert.*

transfor**MER** decoder) dieses um den Coverage Mechanismus (vgl. Abschnitt 2.3.2). CoMER ist ein Transformer-basiertes Modell, welches Teile der originalen Architektur verändert und erweitert. Zusätzlich ist es eines der derzeitigen State-of-the-Art Modelle zur Erkennung von handschriftlichen mathematischen Formeln. Eine Übersicht des CoMER Modells ist in Abbildung 3.1.1 dargestellt. Zu erkennen ist die starke Ähnlichkeit zu der originalen Transformerarchitektur [VSP+17] (vgl. Abbildung 2.4.1).

Zunächst ist der Ersatz der Enkoder-Blöcke durch das DenseNet [ZDD18] zu sehen. Das DenseNet wird zur Merkmalsextraktion der Eingabe verwendet und besteht aus einem CNN, welches Faltungsschichten mit schichtübergreifenden Verbindungen beinhaltet. Dadurch besteht die Eingabe einer Faltungsschicht aus den Ausgaben aller vorherigen Faltungsschichten [ZDD18]. Die Positions-Kodierung wird im CoMER Modell analog zu [VSP+17] realisiert, jedoch ist der Ansatz um eine zweite Dimension erweitert [ZG22a]. Die Dekoder-Blöcke sind dagegen zum größten Teil unverändert. Analog zu der originalen Transformerarchitektur, sind im CoMER Modell zwei Multi-Head-Attention Blöcke zu sehen, wobei der Obere durch das *Attention Refinement Module* (ARM) erweitert wird. Das ARM soll dabei den zuvor vorgestellten Coverage-Mechanismus (vgl. Abschnitt 2.3.2) auf die Transformerarchitektur übertragen. In RNNs kann die Berechnung der Coverage z. B. durch Formel 2.3.4 realisiert werden. Eine naive Übertragung der Formel in die Transformerarchitektur würde jedoch

$\mathcal{O}(TLhd_{attn})$ Speicher für die Coverage-Matrix $\mathbf{F}$ benötigen, bei Multi-Head Attention-Gewichten $\mathbf{A} \in \mathbb{R}^{T \times L \times h}$ und $\mathbf{v}_\alpha \in \mathbb{R}^{d_{attn}}$ (vgl. Formel 2.3.4) [ZG22a]. Zur Reduktion des Speicherbedarfes wird $\mathbf{U}_f \mathbf{f}_i$ außerhalb der tanh Funktion berechnet, wodurch sich

$$\mathbf{e}_t = \tanh(\mathbf{H}_t \mathbf{W}_\alpha + \mathbf{X}\mathbf{U}_\alpha)\mathbf{v}_\alpha + \mathbf{F}_t \mathbf{v}_\alpha \tag{3.1.1}$$

ergibt [ZG22a]. Hierbei wird der gesamte Vektor statt einzelner Elemente berechnet, weswegen eine leicht andere Formulierung der Luong-Attention [LPM15] vorliegt (vgl. Formel 2.3.4). Übertragen auf die Transformerarchitektur soll der Korrekturterm $\mathbf{F}_t \mathbf{v}_\alpha$ die Ausgabe der Multi-Head Attention modifizieren. Das ARM soll diesen Korrekturterm approximieren. Die Eingabe des ARM besteht dabei aus der Ausgabe des Scaled-Dot Product $\mathbf{E} = \mathbf{Q}\mathbf{K}^T / \sqrt{d_k}$ (vgl. Formel 2.4.3) und einer weiteren Attention-Matrix $\mathbf{A}$. Je nach Typ der eingesetzten Coverage wird $\mathbf{A}$ in Abhängigkeit von $\mathbf{E}$ oder der vorherigen Dekoder-Schicht $\mathbf{A}^{j-1}$ berechnet, wobei sich der Korrekturterm zu

$$\hat{\mathbf{E}} = \text{ARM}(\mathbf{E}, \mathbf{A}) = \mathbf{E} - \phi(\mathbf{A}) \tag{3.1.2}$$

$$\phi(\mathbf{A}) = \text{norm}(\max(0, \mathbf{K} * \tilde{\mathbf{C}} + \mathbf{b}_c)\mathbf{W}_c) \tag{3.1.3}$$

ergibt [ZG22a]. $\tilde{\mathbf{C}}$ ist dabei eine umsortierte Matrix mit Zwischensummen der Zeilen aus $\mathbf{A}$. Die Einträge in $\mathbf{C}$ sind daher mit $\mathbf{c}_i = \sum_{k=1}^{i-1} \mathbf{a}_k$ berechenbar. $\mathbf{K}$ bzw. $\mathbf{W}_c$ sind anlernbare Gewichtsmatrizen und $\mathbf{b}_c$ ist ein Bias-Term [ZG22a]. Eine Visualisierung des Korrekturterms in verschiedenen Zeitschritten der Inferenz ist in Abbildung 3.1.1 dargestellt. Dunkle Bereiche werden dabei in einem einzelnen Inferenzschritt weniger stark gewichtet, wohingegen die helleren Bereiche stärker gewichtet werden. Das ARM realisiert daher, analog zur Coverage in RNNs, eine Art Maskierung der Eingabe. In jedem Teilschritt sollen die vorherigen Zeichen weniger stark und die

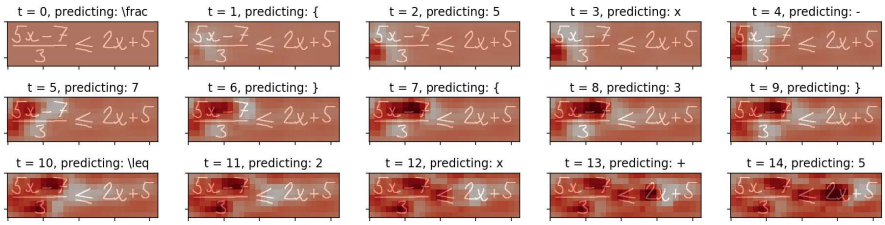


Abbildung 3.1.2: *Visualisierung des approximierten Korrekturterms der Coverage aus [ZG22a]. Je dunkler der Bereich, desto größer ist der Wert in der Korrekturmatrix $\phi(\mathbf{A})$ und umso weniger stark wird der Bereich gewichtet.*

Position des nächsten Zeichens hervorgehoben und stärker gewichtet werden. Wie in Abschnitt 2.4 beschrieben und Abbildung 3.1.1 zu erkennen, besteht der Dekoder im Transformer aus mehreren Blöcken. In jedem Block ist innerhalb der Multi-Head Attention ein ARM Block vorhanden. Zu erkennen ist einerseits die Eingabe der Scaled-Dot Product Attention $\mathbf{E}^j$ und andererseits die benötigten Eingaben zur Berechnung von $\mathbf{A}^j$. In [ZG22a] werden drei Typen der Coverage, die *Self-*, *Cross-* bzw. *Fusion-Coverage*, unterschieden. Die Self-Coverage nutzt dabei direkt die Attention-Gewichte und kann wie folgt definiert werden [ZG22a, VSP+17] (vgl. Abbildung 3.1.1):

$$\mathbf{A}^j = \text{softmax}(\mathbf{E}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}) \tag{3.1.4}$$

Zusätzlich ist in Abbildung 3.1.1 die Ausgabe des ARM nach einer Softmax-Schicht zu erkennen, welche mit dem nachfolgenden Dekoder-Block verbunden ist und mit $\hat{\mathbf{A}}^j = \text{softmax}(\hat{\mathbf{E}}^j)$ (vgl. Formel 3.1.2) berechnet wird. Die Cross-Coverage verwendet die Ausgabe der vorherigen Schicht als Attention-Gewichte $\mathbf{A}$ in Formel 3.1.2, sodass

$$\mathbf{A}^j = \hat{\mathbf{A}}^{j-1} \tag{3.1.5}$$

gilt. Die Fusion-Coverage vereint beide Coverage-Varianten durch Konkatenieren, wodurch sich

$$\mathbf{A}^j = [\text{softmax}(\mathbf{E}^j); \hat{\mathbf{A}}^{j-1}] \tag{3.1.6}$$

ergibt [ZG22a]. Die Fusion-Coverage hat in den Experimenten von [ZG22a] die besten Ergebnisse erzielt. Zudem wird in [ZG22a] gezeigt, dass das ARM mit jeder vorgestellten Art an Coverage eine Verbesserung erzielt und besonders bei langen Sequenzen einen deutlichen Unterschied bewirkt. Dabei decken sich die Ergebnisse aus [ZG22a] mit den Ergebnissen bei der Einführung des Coverage-Terms aus [TLL+16]: Der Attention-Mechanismus wird dahingehend verbessert, dass kein Bereich der Eingabe ausgelassen oder mehrfach betrachtet wird.

Zusammengefasst modifiziert das CoMER Modell die Transformerarchitektur an mehreren Stellen, um es einerseits auf Bilddaten anzupassen und andererseits den Coverage-Mechanismus zu realisieren. CoMER wird in allen Experimenten dieser Arbeit ohne Änderungen an der Architektur mit Fusion-Coverage verwendet.

3.2 SEMI-SUPERVISED LEARNING

Die zuvor behandelten Neuronalen Netze können mithilfe von annotierten Daten und SGD (vgl. Abschnitt 2.1.1) angelernt werden. Wenn ausschließlich annotierte Daten verwendet werden, handelt es sich um vollständig-überwachtes Lernen (Supervised Learning). Im Kontrast dazu werden im vollständig-unüberwachten Lernen keinerlei Annotationen genutzt. Die Cluster- bzw. Strukturanalyse von Daten sind Beispiele des vollständig-unüberwachten Lernens [MGL+18]. Dabei sollen ähnliche Daten in einem Cluster gruppiert werden. Die Varianz innerhalb eines Clusters soll dabei minimal und zwischen Clustern maximal sein [JH10]. So können Ähnlichkeiten in großen Datensätze untersucht werden, z. B. zur Identifikation des Schreibers eines handgeschriebenen Textstückes durch Rückgabe anderer Texte des Schreibers (Writer-Retrieval).
Im semi-überwachten Lernen (Semi-Supervised Learning, SSL) wird die vollständig-überwachte Lernmethodik erweitert, statt diese zu ersetzen. Neben den annotierten Datenpunkten sollen zusätzliche nicht-annotierte Datenpunkte während oder vor dem Lernprozess zur Verbesserung der Erkennungsleistung genutzt werden [EH20]. Der vollständige Datensatz besteht daher aus einem Teil annotierter Daten und einem Teil nicht-annotierter Daten. Bei den nicht-annotierten Daten entfällt die Annotationserstellung, wodurch die Kosten für die Erstellung deutlich niedriger sind.

Viele Ansätze können dabei in zwei Kategorien eingeteilt werden: Das Self-Training und die Konsistenzregularisierung. Die Arbeiten im Bereich der Konsistenzregularisierung wie z. B. [SJT16, LA16, BAP14, XDH+20, LA17, BCG+19, BCC+20, RBH+15] nutzen den Umstand aus, dass die unbekannte Annotation gleich bleibt, auch wenn die Eingabe oder das Netzwerk leicht verändert wird. Intuitiv soll die Vorhersage gleich bleiben, auch wenn die Eingabe z. B. leicht rotiert wird. Dadurch sollen die Datenpunkte ohne Annotationen genutzt werden. Veränderte Eingaben bzw. Netzwerke erzeugen unterschiedliche Ausgaben für einen einzelnen nicht-annotierten Datenpunkt. Die Ausgabeverteilungen sollen aber bei der Konsistenzregularisierung gleich bleiben, da sich die unbekannte Annotation des Datenpunktes nicht verändert. Die Angleichung der Ausgabeverteilungen wird meist über eine separate Loss-Funktion realisiert [RBH+15]. Dadurch soll das Netzwerk konsistente Ausgaben für leichte Variationen der Eingabe bzw. des Netzwerks erzeugen.
Eine Änderung des Netzwerks kann beispielsweise durch Dropout [SHK+14] oder andere Ensemble-Methoden realisiert werden [LA16, BAP14]. Dropout deaktiviert zufällig ausgewählte Gewichte, wohingegen in [BAP14] zufälliges Rauschen auf die Gewichte addiert wird. Methoden wie [LA16] nutzen Zwischenpunkte während des Trainings für die Netzwerkvariation.

Eine Änderung der Eingabe in Form von Bilddaten kann beispielsweise durch zufällige Skalierung, Rotationen oder Translationen realisiert werden [SJT16, XDH+20, LA17, BCG+19, BCC+20, RBH+15]. Damit die Annotation erhalten bleibt, werden die angewendeten Verzerrungen durch vordefinierte Grenzen limitiert. Ansonsten könnte z. B. aus einer 6 durch zu starke Rotation eine 9 oder andersherum werden. Das Anwenden von meist zufallsgesteuerten Verzerrungen auf eine Eingabe wird Augmentierung genannt (vgl. Abschnitt 4.2).
Durch Dropout, Ensembles oder Augmentierung werden so mehrere unterschiedliche Ausgabeverteilungen erzeugt. Das Trainingsziel ist dann die Angleichung der Verteilungen mithilfe einer meist separaten Loss-Funktion [RBH+15].

Beim Self-Training wird die Vorhersage des Netzwerks auf einem nicht-annotierten Datenpunkt zur Erstellung eines 1-aus-k Labels genutzt [L+13]. Dadurch wird ein Pseudolabel im gleichen Format zu den existierenden Annotationen erstellt und zum annotierten Teil hinzugefügt. Im weiterem Trainingsverlauf wird die vergrößerte Menge annotierter Daten zur Verbesserung der Erkennungsleistung verwendet. Dafür wird ein Netzwerk meist mit annotierten Daten vortrainiert und anschließend mithilfe beider Datensätze weiter optimiert. Nach der Konvergenz auf den annotierten Daten wird die erreichte Erkennungsleistung für die Generierung bestmöglicher Vorhersagen genutzt. Diese werden ggf. gefiltert und als Pseudolabel schließlich zum annotierten Teil hinzugefügt. Das Filtern soll dabei die Anzahl an falschen Pseudolabel verringern. Die Erstellung von Pseudolabel kann während des weiteren Trainings wiederholt werden, um stetig neue bzw. verbesserte Pseudolabel zu generieren. Durch die erhöhte Varianz sowie die Möglichkeit, dass die Vorhersage des Pseudolabel korrekt ist, soll die Erkennungsleistung gesteigert werden. Beide Trainingsschritte können auch zusammengelegt werden, wodurch bereits zu Beginn des Trainings Pseudolabel erzeugt werden. Da diese zu Beginn meist unsicher bzw. falsch sind, kann dies das Training auch behindern [RHS05].

Ansätze wie [ZOKB19, WKLQ21, PDXL21, SBL+20, XSY+21] kombinieren dabei das Self-Training und die Konsistenzregularisierung. In [ZOKB19] werden z. B. Rotationen, und in [WKLQ21] Ensembles verwendet. Zwei Netze werden in [PDXL21] zusammen genutzt, indem das erste Netz Pseudolabel generiert und von der Erkennungsleistung des Zweiten beeinflusst wird. Das zweite Netz wird dabei mit den generierten Pseudolabel angelernt und mithilfe der annotierten Daten evaluiert. In [SBL+20] wird ein konstanter Schwellenwert zur Filterung von Pseudolabel und Augmentierung zur Konsistenzregularisierung eingesetzt. Darauf aufbauend verwendet [XSY+21] einen dynamischen Schwellenwert, welcher mit fortschreitendem Training erhöht wird. Da-

hingegen werden in [ZWH+21] klassenspezifische Schwellenwerte verwendet, welche von der Erkennungsleistung einer Klasse und einem vordefinierten Schwellenwert abhängen. Ein ähnliches Vorgehen wird auch in [WCH+23] verwendet, wobei der Schwellenwert ausschließlich adaptiv, basierend auf der Erkennungsleistung und pro Klasse, bestimmt wird. In [CTF+23] wird kein Schwellenwert verwendet, sondern stattdessen das Gewicht im Training reduziert, wenn die Vorhersage unsicher ist bzw. mit einer niedrigen Konfidenz generiert wird. Die Evaluation hat dabei gezeigt, dass die größere Varianz im Training für die Erkennungsleistung sehr wichtig ist [CTF+23].

Weitere Methoden wie [LXH21] nutzen mehrere Trainingsziele: die Generierung von Pseudolabel und die Einbettung in einen niedrigdimensionalen Vektor. Durch einen Contrastive-Loss [GH10] werden die beiden Ziele voneinander beeinflusst. Der Contrastive-Loss soll dabei die Distanz zwischen Datenpunkten einer Klasse und Datenpunkten aller anderen Klassen maximieren [KTW+20]. Andere Ansätze wie [DSRB14] verwenden die nicht-annotierten Datenpunkte in einem Vortraining. Dabei wird jeder Datenpunkt als eigene Klasse angesehen, wobei Augmentierungen während des Vortraining verwendet werden. Anschließend wird die letzte Schicht des Netzwerks mit den annotierten Daten erneut angelernt, wobei die vorherigen Schichten ihre Gewichte beibehalten.

Wie zu erkennen ist, sind die Ansätze vielseitig. Zudem geht der Trend in den State-of-the-Art Ansätzen hin zur Erweiterung der FixMatch [SBL+20] Methodik [XSY+21, CTF+23, ZWH+21, WCH+23, ZYH+22, LXH21, WLM+22]. Die Ansätze probieren meist bessere Pseudolabel zu generieren oder diese geschickt zu filtern, sodass viele und gleichzeitig möglichst richtige Pseudolabel in das Training aufgenommen werden [RDRS21, PAQ23].

3.2.1 *FixMatch*

FixMatch [SBL+20] ist dabei eine der ersten Methoden, welche die Konsistenzregularisierung durch starke und schwache Augmentierungen und die Erstellung von Pseudolabel im Self-Training miteinander vereint [WCF+22]. Die Grundidee der FixMatch Methodik wird innerhalb dieser Arbeit auf die Vorhersage von Sequenzen übertragen und darüber hinaus erweitert (vgl. Abschnitt 4.3.3).
Für einen nicht-annotierten Datenpunkt wird eine schwach und eine stark augmentierte Variante erzeugt. Die Veränderungen des Eingabebildes sind bei der Anwendung der starken Augmentierung deutlich ausgeprägter. In FixMatch wird dafür RandAug

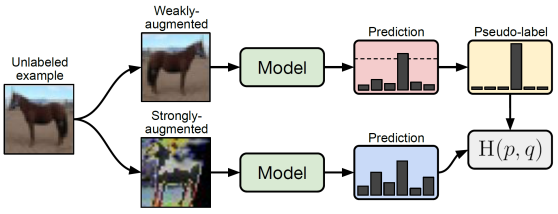


Abbildung 3.2.1: *Visualisierung der einzelnen Schritte der FixMatch Methodik aus [SBL+20].*

[CZSL20] bzw. CTAugment [BCC+20] (vgl. Abschnitt 4.2.2) genutzt, wodurch mehrere zufällige Verzerrungen angewendet werden, die im Falle von CTAugment anlernbar sind. Der Konfidenzwert der Vorhersage auf einer schwach augmentierten Variante wird dann mithilfe eines zuvor festgelegten Schwellenwert τ genutzt, um ein 1-aus-k Label zu erzeugen.
Daraufhin wird das Pseudolabel und die stark augmentierte Variante für das Training genutzt. Dadurch wird ähnlich zu den bereits vorgestellten Methoden die Konsistenzregularisierung realisiert. Die Ausgabe des Netzwerks soll für die schwach und stark augmentierte Variante gleich sein. Im Gegensatz zu den „Vorgängern" Mix- und ReMixMatch [BCG+19, BCC+20] und vielen anderen Methoden der Konsistenzregularisierung [SJT16, LA16, BAP14, XDH+20, LA17, RBH+15], wird dabei jedoch keine separate Loss-Funktion für die nicht-annotierten Datenpunkte verwendet.
Das beschriebene Vorgehen ist in Abbildung 3.2.1 dargestellt und zeigt die modifizierbaren Bestandteile der Methode: die konkrete Implementierung der schwachen und starken Augmentierung, sowie der Schwellenwert τ, welcher für die Generierung von 1-aus-k Pseudolabel genutzt wird.

3.3 KONFIDENZBEWERTUNG

Der Konfidenzwert ist bei der FixMatch Methodik ein elementarer Bestandteil, da auf dessen Grundlage ein Label in das Training aufgenommen oder ausgelassen wird. Der Konfidenzwert soll dabei die Unsicherheit bzw. Sicherheit einer Vorhersage zusammenfassen. Wie zuvor beschrieben, kann die Ausgabe der Softmax-Schicht als Pseudowahrscheinlichkeit einer Vorhersage angesehen werden (vgl. Abschnitt 2.1). Bei einem 1-aus-k Pseudolabel kann dann der entsprechende Wert der Softmax-Funktion als Konfidenzwert verstanden werden. Da dieser Wert in der Regel jedoch nicht repräsentativ für die tatsächliche Erkennungsleistung ist, handelt es sich um eine Pseudowahrscheinlichkeit. Dahingegen entspricht der Konfidenzwert eines kalibrierten Netzwerks annähernd der Wirklichkeit, sodass z. B. von 100 Vorhersagen mit einer

Konfidenz von 0.8 idealerweise 80 korrekt sind. Das Ziel der Kalibrierung ist es, eine möglichst realitätsnahe Pseudowahrscheinlichkeit als Konfidenzwert auszugeben. Bei den Vorhersagen von Sequenzen muss das Konfidenzmaß die einzelnen Softmax-Verteilungen aller Zeichen aggregieren. Analog kann danach die Kalibrierung der Gesamtvorhersage betrachtet werden, sodass von 100 Sequenzen mit einer Konfidenz von 0.8 idealerweise 80 vollständig korrekt sein sollen.

Eine gute Kalibrierung ist für die Fehlervermeidung bei der Auswahl von Pseudolabel sehr wichtig [RDRS21]. Nach [RDRS21] ist das reine Filtern auf der Grundlage eines unkalibrierten Konfidenzwertes, wie in der FixMatch Methodik, deutlich verbesserbar. Wie in [RDRS21, PAQ23] gezeigt, erreichen kalibrierte Modelle bessere Erkennungsleistungen, indem die kalibrierte Unsicherheit bei der Auswahl der Pseudolabel berücksichtigt wird. Darüber hinaus bildet der Konfidenzwert die Eingabe weiterführender Methoden wie das *Partial Labeling* (vgl. Abschnitt 4.3.3) oder das Gewichten der Datenpunkte aus [CTF+23]. Eine bessere Kalibrierung erzielt dabei auch bessere Ergebnisse (vgl. Abschnitt 5.5 und [CTF+23, RDRS21]).
Daher spiegelt die Konfidenz des Neuronalen Netzes idealerweise die Realität wider, sodass hohe Konfidenzwerte mit vielen fehlerfreien Hypothesen korrelieren und niedrige Konfidenzwerte für unsichere und fehlerbehaftete Hypothesen sprechen. Jedoch neigen Neuronale Netze basierend auf ReLU Aktivierungsfunktionen zur *Overconfidence* [HAB19, KHH20b, KHH20a, GPSW17]. Intuitiv sind damit hohe Konfidenzwerte bei Daten mit großer Distanz zu den Trainingsdaten gemeint. Im Kontext der Zeichen- bzw. Sequenzerkennung könnte dies z. B. ein im Alphabet der Trainingsdaten nicht vorkommendes Zeichen sein, welches vom Netzwerk mit hoher Konfidenz mit einem anderen Zeichen verwechselt wird. Ein deutlich anderer Schreibstil könnte aber ebenso zu Overconfidence führen. Neben der ReLU Aktivierungsfunktion werden in [GPSW17] weitere mögliche Gründe, wie die Tiefe sowie Breite eines Neuronalen Netzwerks und Batch Normalization [IS15], identifiziert. Die Ansätze zur Verbesserung der Kalibrierung sind erneut vielfältig. Viele Ansätze verwenden Nachbearbeitungsmethoden, welche die Ausgabe des Netzes (ggf. vor der Softmax-Schicht) verändern. Andere Ansätze nutzen z. B. die Modellgewichte oder eine andere Lernmethodik.

Binning Methoden wie [ZE01, ZE02, NCH15] teilen den Konfidenzraum in disjunkte Intervalle und nutzen einen Validierungsdatensatz zur Bestimmung kalibrierter Konfidenzwerte für jedes Intervall. Der Validierungsdatensatz ist dabei eine Teilmenge des annotierten Datensatzes, welcher nicht für das Training verwendet wird. Zur Bestimmung der Intervallgrenzen werden feste Regeln [ZE01], Optimierungsprobleme [ZE02] oder bayesche Modelle [NCH15] verwendet.

Transformationsfunktionen wie Platt Scaling [P+99] nutzen im Kontext von Neuronalen Netzen eine anlernbare Funktion $\hat{\mathbf{y}} = \text{softmax}(\alpha\mathbf{y} + \beta)$ zur Transformation der Ausgabe des Netzwerks $\mathbf{y}$, bevor die Softmax-Funktion angewendet wird. Die Parameter α und β können dabei mithilfe des SGDs auf dem Validierungsdatensatz angelernt werden [NMC05, GPSW17]. Vektor bzw. Matrix Scaling verwenden für α eine Gewichtsmatrix $\mathbf{W}_\alpha$, welche im Falle von Vektor Scaling eine Diagonalmatrix ist, und für β einen anlernbaren Vektor [GPSW17].

All-Layer bzw. Last-Layer-Laplace-Approximation [KHH20a] nutzen die Laplace Approximation [Wal69] zur Bestimmung des kalibrierten Konfidenzwertes. Dabei werden die Modellparameter mithilfe von Normalverteilungen approximiert. Das Vorgehen zur Parameterbestimmung der Normalverteilung sowie die Auswahl an Modellparametern kann dabei variiert werden. Dadurch ändert sich der Berechnungsaufwand und die Kalibrierungsleistung [DKI+21]. Das approximierte Ersatznetzwerk f_* wird dann zur Berechnung des Konfidenzwertes genutzt, indem die Pseudowahrscheinlichkeit beispielsweise mithilfe der Monte Carlo Integration [Caf98] über die *likelihood* $\int p(\mathbf{y}|f_*)p(f_*|x_*, \mathcal{D})d\Theta$ bestimmt wird [KHH20a, Wal69, DKI+21].

3.3.1 *Temperature Scaling*

Das Temperature Scaling ist ein vereinfachter Fall des Platt Scalings [P+99] und damit eine Nachbearbeitungsmethode. Dabei wird nur der α Parameter optimiert, wobei die zu optimierende Funktion leicht abgeändert mit

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{y}/T) \tag{3.3.1}$$

definiert ist [GPSW17]. Trotz der im Vergleich zu anderen Binning bzw. Scaling Methoden einfachen Formulierung, erreicht es in der Übersichtsarbeit [GPSW17] die besten Kalibrierungsergebnisse. Auch im Vergleich zur Laplace-Approximation erzielt Temperature Scaling vergleichbare Kalibrierungsleistungen auf verschiedenen Datensätzen [DKI+21].
Motiviert durch die kompetitiven Kalibrierungsergebnisse in Kombination mit der einfachen Berechnung sowie Optimierung wird Temperature Scaling in den Experimenten dieser Arbeit eingesetzt (vgl. Abschnitt 4.3.2 und 5.5).

3.3.2 *LogitNorm*

Beim Temperature Scaling wird, wie in Formel 3.3.1 zu erkennen, die Ausgabe des Netzwerks unabhängig der Eingabe mit dem Parameter T skaliert. Im Kontrast dazu wird in LogitNorm [WXC+22] der Ausgabevektor auf eine bestimmte Länge normiert. Dadurch ergibt sich eine Skalierung der Ausgabe in Abhängigkeit von der Eingabe [WXC+22]. Die Skalierung der Ausgabe ist mit

$$\hat{\mathbf{y}} = \frac{\mathbf{y}}{\tau \cdot |\mathbf{y}|} \tag{3.3.2}$$

gegeben [WXC+22]. Zu erkennen ist eine Normierung des Ausgabevektors auf eine Länge von τ^{-1}. Das Besondere an dieser Methode ist die Normierung während des Trainings und nicht während der Inferenz. Die vorherigen Methoden beschreiben einen Nachbearbeitungsschritt der Netzwerkausgaben bzw. der Schätzung der Gewichte während der Inferenz. Dahingegen wird die Normierung in LogitNorm ausschließlich vor dem Anwenden der Loss-Funktion eingesetzt, wodurch sich die Gewichte entsprechend des normierten Vektors verändern. Nach [WXC+22] wird dadurch ein kalibriertes Netz während des Trainings erreicht, welches in Kombination mit Temperature Scaling bessere Ergebnisse erzielt als Temperature Scaling alleine.
In der Praxis wird LogitNorm durch Anwenden von Formel 3.3.2 vor dem Cross-Entropy-Loss realisiert (vgl. Abschnitt 2.1), wobei in der Inferenz lediglich Temperature Scaling eingesetzt wird. Die LogitNorm Methodik wird für einen Vergleich in den Experimenten zur Kalibrierung angewendet (vgl. Abschnitt 5.5).

4

METHODIK

Nach dem Überblick über die hauptsächlichen Bereiche dieser Arbeit und deren Ansätze, werden im Folgenden die konkreten Methodiken sowie Konfidenzmaße näher beschrieben. Damit sollen die beiden zentralen Forschungsfragen beantwortet werden: In welchem Maß können nicht-annotierte Daten die Erkennungsleistung eines Neuronalen Netzes zur handschriftlichen mathematischen Formelerkennung verbessern und inwieweit kann der annotierte Datenaufwand reduziert werden.

Zur Untersuchung und Beantwortung der Forschungsfragen wird das CoMER Modell (vgl. Abschnitt 3.1.1) für die bi-direktionale Erkennung von handschriftlichen mathematischen Formeln genutzt. Das Modell wird zum Einen vollständig-überwacht angelernt, indem handschriftlich geschriebene mathematische Formeln aus verschiedenen Datensätzen oder synthetisch generierte Daten verwendet werden. Zum Anderen wird das Netzwerk semi-überwacht angelernt, indem das Self-Training auf die Sequenzvorhersage übertragen und erweitert wird. Dabei werden Pseudolabel für nicht-annotierte Datenpunkte bestimmt und zu der annotierten Teilmenge hinzugefügt. Die entwickelte Self-Training Methode orientiert sich dabei in manchen Teilen an der FixMatch Methodik. Die Konsistenzregularisierung wird analog mithilfe von RandAug (vgl. Abschnitt 4.2.2) realisiert. Jedoch wird die Konfidenzbewertung auf Sequenzen ausgeweitet und verschiedene Konfidenzmaße sowie Kalibrierungsmethoden untersucht (vgl. Abschnitt 4.3.2). Darüber hinaus wird die Filterung von Sequenzen mithilfe von Partial Labeling aufgeweicht (vgl. Abschnitt 4.3.3). Zusätzlich werden synthetisch generierte Daten in einem Vortraining verwendet (vgl. Abschnitt 4.1.1). Insgesamt sollen die nicht-annotierten Datenpunkte die Erkennungsleistung des vollständig-überwacht angelernten Netzwerks verbessern.
Die resultierende Self-Training Methode ist durch die Änderungen und Erweiterungen deutlich von FixMatch zu unterscheiden. Der beschriebene Ablauf ist nochmals in Abbildung 4.0.1 dargestellt und zeigt die Trainingsschleifen im vollständig-überwachten sowie semi-überwachten Lernen.

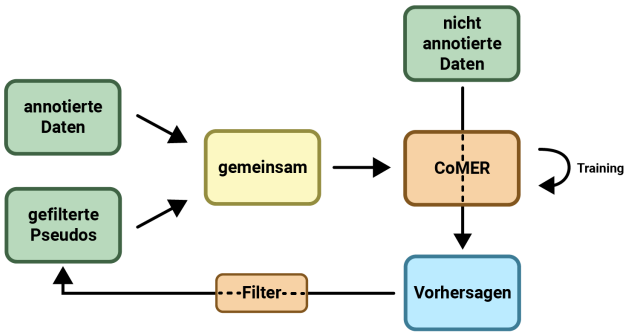


Abbildung 4.0.1: *Übersicht der allgemeinen Self-Training Methode. Die annotierten bzw. Schwellenwert-passierenden nicht-annotierten Daten werden für das Training des CoMER [ZG22a] Modells genutzt, welches Vorhersagen über die gesamten nicht-annotierten Daten generiert, um die nächste Menge an gefilterten Pseudolabel zu erzeugen. Die annotierten Daten können dabei auch synthetisch generiert werden.*

4.1 SUPERVISED LEARNING

Für das Training des CoMER Modells mit annotierten Datenpunkten bzw. Pseudolabel wird SGD mit dem Cross-Entropy Loss ϵ_{CE} verwendet (vgl. Abschnitt 2.1.1). Bei der Verwendung von LogitNorm wird die Ausgabe vor dem Cross-Entropy Loss entsprechend Formel 3.3.2 normiert. Wenn die gesamte Menge an annotierten Daten in Minibatches der Größe 8 für die Optimierung der Gewichte genutzt wird, entspricht dies einer Epoche. Für das vollständig-überwachte Anlernen werden 300 Epochen verwendet. Falls jedoch weniger als 20% der annotierten Daten für das Training genutzt werden, werden 450 Epochen verwendet, damit eine vollständige Konvergenz erreicht wird. Die Lernrate ist dabei, analog zu der Referenzimplementierung von CoMER [ZG22b], anfänglich auf $\lambda_{start} = 0.08$ gesetzt. Es wird ein *Step-Scheduler* zur Verringerung der Lernrate in $c_{steps} = 8$ Schritten auf $\lambda_{target} = 8 \cdot 10^{-5}$ eingesetzt. Dieser bewirkt, dass alle $\lceil 300/(8+1) \rceil = 34$ Epochen die Lernrate um einen Faktor γ reduziert wird, sodass in den letzten 34 Epochen $\lambda = 8 \cdot 10^{-5}$ gilt. Zur Berechnung wird

$$\gamma = e^{\frac{\ln(\lambda_{target} \cdot \lambda_{start}^{-1})}{c_{steps}}} \tag{4.1.1}$$

verwendet. Während der Inferenz auf dem Validierungs- bzw. Testdatensatz wird eine Minibatch der Größe 4 verwendet, da durch die bi-direktionale Beam-Search mit Beam-Breite 10 mehr Grafikspeicher benötigt wird als beim Training. Mit der reduzierten Batch-Größe wird das Limit des Grafikspeichers nicht überschritten. Dabei werden

die Hypothesen parallel von beiden Seiten erzeugt. Die bzgl. dem *ORI* Konfidenzmaß (vgl. Abschnitt 4.3.2) beste Hypothese wird dann als Vorhersage für den Datenpunkt genutzt.

4.1.1 *(Vor-)Trainieren mit synthetischen Daten*

Neben den verwendeten Datensätzen mit handgeschriebenen mathematischen Formeln werden synthetisch generierte Daten in einem Vortraining verwendet. Analog zu [SRN17] ist es das Ziel, eine Art implizites Sprachmodell vor dem eigentlichen Training zu erlernen. Dabei soll die Syntax von allgemein wohlgeformten LaTeX Sequenzen erlernt werden, sodass die ausgegebene Sequenz tendenziell syntaktisch korrekt ist. Nach [SRN17] kann ein implizites Sprachmodell in überparametrisierten Modellen beobachtet werden, zu welchen das CoMER Modell mit 6.4 Millionen Parametern vermutlich zählt.

Zur Erzeugung eines synthetischen Datensatzes wird eine LaTeX Sequenz mithilfe eines Renderers zu einem Bild konvertiert. Dabei können weitere Vorverarbeitungsschritte genutzt werden, damit die Formel im Bildformat ähnlich zu den realen Trainingsdaten ist. Im Falle des CROHME Datensatzes wird die Formel z. B. in weiß auf schwarzem Hintergrund gerendert. Durch ein solches Vorgehen ist es möglich eine große Menge an Trainingsdaten zu generieren, da die LaTeX Sequenzen mithilfe einer Grammatik prozedural erzeugt werden könnten. Bei der prozeduralen Generierung werden Grundregeln zur Struktur einer Formel in Kombination mit zufallsgesteuerter Auswahl genutzt, um prinzipiell unendlich viele Sequenzen zu erzeugen. Alternativ können öffentlich zugängliche Datensätze, welche LaTeX Sequenzen beinhalten, für die Erstellung genutzt werden. Unabhängig von der Quelle der LaTeX Sequenzen ist keine Annotation der Datenpunkte erforderlich, da die Bilder auf Grundlage der Annotation erstellt werden. Das Netzwerk wird im Vortraining dabei analog zum vollständig-überwachten Training angelernt. Je nach Größe des Datensatzes sind jedoch ggf. weniger Epochen für eine Konvergenz erforderlich (vgl. Abschnitt 5.1.2).

4.2 DATENAUGMENTIERUNG

Die annotierten Datenpunkte aus dem normalen oder synthetischen Datensatz werden vor der Eingabe in das Netzwerk augmentiert. Wie zuvor bereits kurz beschrieben, ist mit der Augmentierung das Verzerren der Eingabe gemeint. Die Anwendung der

Augmentierungsfunktion ϕ kann als stochastischer Prozess angesehen werden. Wird ϕ mehrfach auf die gleiche Eingabe angewandt, sind die Ausgaben der Funktion im Allgemeinen ungleich, sodass $\phi(\mathbf{x}) \neq \phi(\mathbf{x})$ gilt. Zufallsgesteuert wird die Eingabe entsprechend meist vordefinierter Transformationen modifiziert. Dabei können auch mehrere Transformationen hintereinander angewendet werden. Die Augmentierung kann sowohl *Online* als auch *Offline* durchgeführt werden. Die Offline Augmentierung beschreibt einen einzigen Vorverarbeitungsschritt der Trainingsdaten. Die Verzerrungen werden dabei vor dem Beginn des Trainings durchgeführt, sodass mehrere Varianten eines Datenpunktes erzeugt werden. In der Online Augmentierung werden die Verzerrungen fortlaufend während des Trainings durchgeführt. Vor dem Beginn eines einzelnen Trainingsschrittes werden die Trainingsdaten einer Minibatch entsprechend der Augmentierungsfunktion verzerrt. Dadurch können viele Varianten während des Trainings verwendet werden, wobei sich die Trainingsdauer durch den gesteigerten Berechnungsaufwand erhöht.

Mithilfe der Augmentierung kann die Variation im Datensatz erhöht werden, ohne eine größere Menge Daten zu sammeln und zu annotieren. Ein einzelner annotierter Datenpunkt kann, je nach Augmentierungsfunktion, zu unbegrenzt vielen Variationen modifiziert werden. Dadurch wird die Varianz für eine einzelne Annotation erhöht. Da jedoch auch eine Varianz der Annotationen selber benötigt wird, löst die Augmentierung nicht den Bedarf an eine möglichst große Menge Daten. Durch die Augmentierung soll der approximierte Klassifikator jedoch besser zwischen den Datenpunkten des Datensatzes generalisieren [SK19, XYFP23].

4.2.1 *ScaleAug*

Die Autoren des CoMER Modells [ZG22a] (vgl. Abschnitt 3.1.1) nutzen für das Training die Augmentierungsfunktion ScaleAug [LJLZ20]. Das Eingabebild wird dabei zufällig innerhalb vordefinierter Grenzen skaliert. Die Funktion ϕ kann durch

$$\phi(\mathbf{X}) = \text{resize}(\mathbf{X}, kW_x, kW_y), k \in [S_{min}, S_{max}] \tag{4.2.1}$$

definiert werden, wobei *resize* die lineare Interpolation zum Hoch- bzw. Herunterskalieren des Eingabebildes $\mathbf{X}$ nutzt. Für jede Anwendung der Funktion wird k uniform aus dem Intervall $[S_{min}, S_{max}]$ gezogen. W_x bzw. W_y entsprechen jeweils den Längen der ersten beiden Dimensionen des Eingabebildes (Länge und Breite).
In [ZG22a] hat alleinig das Hinzufügen der ScaleAug während des Trainings des BTTR Modells [ZGY+21] eine deutliche Verbesserung erzielt, welche jedoch hinter

den Verbesserung der Autoren bleibt [LJLZ20]. ScaleAug wird, analog zu CoMER [ZG22a], in allen Experimenten dieser Arbeit, während des Trainings auf annotierten Datenpunkten und für die schwache Augmentierung (vgl. Abschnitt 3.2.1), verwendet.

4.2.2 *RandAug*

Für die starke Augmentierung der Konsistenzregularisierung wird RandAug [CZSL20] verwendet (vgl. Abschnitt 3.2.1). Eine feste Menge von Transformationen ist dabei in der RandAug Methode definiert. Eine gegebene Anzahl N von Transformationen wird dann aus der Menge T_{full} gezogen und nacheinander auf die Eingabe mit einer Stärke M angewandt [CZSL20]. Formal kann RandAug daher wie folgt als rekursive Funktion definiert werden:

$$\begin{aligned}
\phi(\mathbf{X}) &= \text{RandAug}(\mathbf{X}, N, M) \\
\text{RandAug}(\mathbf{X}, N, M) &= \text{SingleAug}(\mathbf{X}, N, M, T_{\text{full}}) \\
\text{SingleAug}(\mathbf{X}, 0, M, T) &= \mathbf{X} \\
\text{SingleAug}(\mathbf{X}, N, M, \emptyset) &= \mathbf{X} \\
\text{SingleAug}(\mathbf{X}, N, M, T) &= \text{SingleAug}(t(\mathbf{X}), N-1, M, T \setminus t), t \in T
\end{aligned} \tag{4.2.2}$$

In der zweiten Zeile wird die Rekursion durch den Aufruf von SingleAug mit der vollständigen Menge an Transformationen T_{full} gestartet. Darauf folgend sind die beiden Rekursionsanker zu erkennen, welche die Eingabe auf sich selbst abbilden, wenn keine weiteren Transformationen verfügbar sind oder die geforderte Anzahl an Transformationen angewendet wurde. In der letzten Zeile ist die eigentliche Rekursion zu sehen, in welcher eine einzelne Transformation t zufällig aus der verfügbaren Menge T gezogen wird und auf die Eingabe der Funktion angewendet wird. Mit der

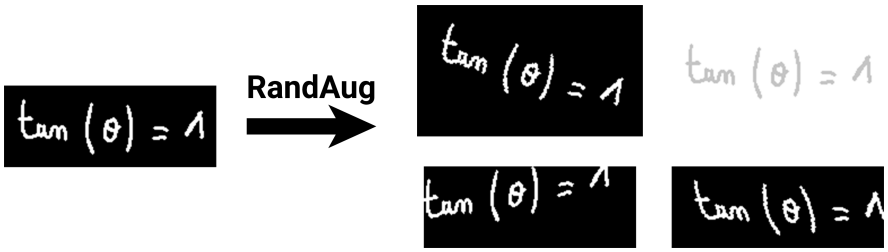


Abbildung 4.2.1: *Beispiele mehrerer Augmentierungen mit RandAug, N = 3 und den Transformationen aus Tabelle A.1.1, angewandt auf ein Bild des CROHME [MZM+19] Datensatzes.*

modifizierten Eingabe, der reduzierten Transformationsliste sowie der dekrementierten Zahl an übrigen Transformationen, wird die Rekursion bis zu N-mal fortgesetzt. Beispiele möglicher Ergebnisse sind in Abbildung 4.2.1 dargestellt und zeigen mehrere unabhängige Aufrufe der RandAug Funktion auf ein Eingabebild.

Wie in FixMatch [SBL+20], wird die Stärke M mit einem zufälligen Wert aus einem vordefinierten Intervall ersetzt. Dadurch wird die Stärke der Transformation bei jeder Anwendung uniform aus einem definierten Intervall gezogen. Die vollständige Liste an verwendeten Transformationen inklusive des Stärke-Intervalls ist in Tabelle A.1.1 dargestellt.

4.3 SEMI-SUPERVISED LEARNING

Nach dem vollständig-überwachten Anlernen des Netzwerks, wird der beste Zwischenstand (bzgl. der fehlerfreien Erkennungsleistung auf dem Validierungsdatensatz) für das Self-Training verwendet. Innerhalb von 250 Epochen, $c_{steps} = 5$, $\lambda_{start} = 0.01$ und $\lambda_{target} = 8 \cdot 10^{-5}$, wird die semi-überwachte Methodik des jeweiligen Experiments durchgeführt.

4.3.1 *Self-Training*

Das Self-Training soll Pseudolabel für nicht-annotierte Datenpunkte erzeugen. In einem zweiten Training sollen annotierte und nicht-annotierte Datenpunkte zusammen für die Optimierung des CoMER Modells verwendet werden (vgl. Abbildung 4.0.1). Die Sequenzen der Pseudolabel werden mithilfe der Beam-Search bestimmt (vgl. Abschnitt 2.5). Der Konfidenzwert wird mithilfe eines Konfidenzmaßes berechnet, wobei verschiedene Maße evaluiert werden (vgl. Abschnitt 4.3.2). Die Filterung von unsicheren Hypothesen wird zunächst anhand eines festgelegten Schwellenwertes durchgeführt und später mit der Partial Labeling Heuristik relaxiert (vgl. Abschnitt 4.3.3). Dadurch werden nicht-annotierte Datenpunkte mit einer genügend hohen Konfidenz in das Training aufgenommen, indem die Datenpunkte und die Pseudolabel in einem Zwischenspeicher abgelegt werden. Vor einer Trainingsepoche werden Minibatches aus allen annotierten Datenpunkten sowie nicht-annotierten Datenpunkten mit Pseudolabel erstellt. Die Datenpunkte mit Pseudolabel werden dabei gleichmäßig auf alle Minibatches verteilt, sodass immer ein Teil annotierter und ein Teil nicht-annotierter Daten in einer Minibatch vorkommen. Dadurch kann die Minibatch-Größe konstant vorgegeben werden und das Verhältnis von annotierten zu nicht-annotierten

Datenpunkten entspricht in etwa dem Verhältnis der Datensatzgrößen. Zusätzlich wird, analog zu FixMatch [SBL+20], die Konsistenzregularisierung mit RandAug für die starke Augmentierung und ScaleAug für die schwache Augmentierung realisiert (vgl. Abschnitt 4.2). Dadurch werden Datenpunkte mit einem Pseudolabel in der Trainingsschleife stark augmentiert, wobei annotierte Datenpunkte nur durch ScaleAug transformiert werden.
Im Gegensatz zu FixMatch wird das Pseudolabel eines nicht-annotierten Datenpunktes nicht während der Trainingsschleife bestimmt. Einerseits aufgrund der Grafikspeicherlimitierung und andererseits zur Vermeidung eines ständigen Wechsels zwischen dem Trainings- und Inferenzbetrieb des Netzwerks. Da während der Inferenz keine Gradienten bestimmt werden und die Gewichte unverändert bleiben sollen, müsste das Netzwerk in jeder Trainingsiteration — ggf. mehrmals — zwischen den beiden Modi wechseln. Um dies zu vermeiden, werden die Pseudolabel als zusätzlicher Schritt nach der Validierung bestimmt. Dies geschieht nach jeder zweiten Epoche, sodass sich die Schleife *Train* → *Train* → *Validate* → *Pseudolabel* ergibt.
Im *Train*-Schritt werden die annotierten Daten, wie in Abschnitt 4.1 beschrieben, zum Anlernen verwendet. Falls Pseudolabel im Zwischenspeicher vorliegen, werden diese ebenfalls genutzt. Dadurch besteht, abgesehen von der starken Augmentierung, kein Unterschied zwischen den annotierten und nicht-annotierten Daten mit Pseudolabel während des Trainings. Im *Validate*-Schritt wird zum Einen die fehlerfreie Erkennungsrate auf dem Validierungsdatensatz bestimmt, damit der beste Zwischenstand des Netzwerks abgespeichert werden kann. Zum Anderen werden die Inferenzergebnisse für die Optimierung des Temperature Scaling Parameters T genutzt (vgl. Abschnitt 4.3.2). Während des *Pseudolabel*-Schritts werden Hypothesen mithilfe der Beam-Search (vgl. Abschnitt 2.5) für alle nicht-annotierten Datenpunkte bestimmt. Dabei werden die Pruning Methoden aus Abschnitt 2.5.1 mit $\lambda_{\text{abs}} = 5$, $\lambda_{\text{rel}} = 2$, $\lambda_{\text{local}} = 2$ und maximal 5 direkten Nachfolgern einer unfertigen Hypothese genutzt. Anschließend wird der Konfidenzwert mithilfe des jeweiligen Konfidenzmaßes (vgl. Abschnitt 4.3.2) bestimmt und für das Filtern bzw. Partial Labeling verwendet. Liegt der Konfidenzwert oberhalb des Schwellenwertes, so wird die Hypothese als Pseudolabel für den nicht-annotierten Datenpunkt genutzt und in einem Zwischenspeicher abgelegt. Unterhalb des Schwellenwertes wird die Hypothese verworfen. Vor Beginn des *Train*-Schritts wird der Zwischenspeicher zum Erstellen der größeren Menge an Batches verwendet, indem die Schwellenwert-passierenden Pseudolabel gleichmäßig über alle Minibatches verteilt werden.

4.3.2 *Konfidenzmaße und Kalibrierung*

Für die Filterung unsicherer Hypothesen wird ein Konfidenzmaß benötigt. In diesem müssen die Vorhersagen der einzelnen Zeichen einer Hypothese aggregiert bzw. zusammengefasst werden. Das Konfidenzmaß reduziert also N Pseudowahrscheinlichkeiten auf einen einzelnen Wert. Die Autoren des CoMER Modells [ZG22a] nutzen ein Konfidenzmaß im letzten Schritt der Beam-Search. Nachdem B Hypothesen durch die Beam-Search gefunden wurden, wird der Cross-Entropy Loss der anderen Schreibrichtung von der durchschnittlichen log-Pseudowahrscheinlichkeit abgezogen. Der daraus berechnete Konfidenzwert wird zur Auswahl der besten Hypothese aus den B Kandidaten verwendet, indem die Hypothese mit der maximalen Konfidenz ausgegeben wird. Formal ergibt sich das „originale" Konfidenzmaß (*ORI*) einer Hypothese $(x_1, ..., x_n)$ mit log-Pseudowahrscheinlichkeiten $(y_1, ..., y_n)$ in der Richtung der Hypothese sowie $(\hat{y}_1, ..., \hat{y}_n)$ in Rückrichtung und Alphabet-Verteilungen $(\hat{\mathbf{y}}_1, ..., \hat{\mathbf{y}}_n)$ zu:

$$ORI(x_1, ..., x_n, y_1, ..., y_n, \hat{y}_1, ..., \hat{y}_n, \hat{\mathbf{y}}_1, ..., \hat{\mathbf{y}}_n) = \frac{\sum y_i - \epsilon_{CE}(\hat{\mathbf{y}}_i, x_i)}{n} \tag{4.3.1}$$

Dabei kann die log-Pseudowahrscheinlichkeit eines einzelnen Zeichens x_i mithilfe der Softmax-Funktion aus der Verteilung über dem gesamten Alphabet $\mathbf{y}_i$ und $y_i = \ln(\text{softmax}(\mathbf{y}_i)_{(x_i)})$ berechnet werden. Die x_i-te Komponente entspricht dann der log-Pseudowahrscheinlichkeit des Zeichens x_i. Durch $\epsilon_{CE}(\hat{\mathbf{y}}_i, x_i)$ wird der Cross-Entropy Loss einer einzelnen Alphabet-Verteilung eines Zeichens der Rückrichtung berechnet (vgl. Formel 2.1.4). Ein zusätzlicher Inferenzschritt wird dabei zur Bestimmung von $(\hat{\mathbf{y}}_1, ..., \hat{\mathbf{y}}_n)$ verwendet, wobei alle B Hypothesen gespiegelt werden. So wird beispielsweise aus $(\text{SOS}, x_1, x_2, \text{EOS}) \rightarrow (\text{EOS}, x_2, x_1, \text{SOS})$. Für die Berechnung der Cross-Entropy wird dann die entsprechende Verteilung der Rückrichtung verwendet. Auf das Beispiel bezogen wird für $\hat{\mathbf{y}}_1$ (korrespondierend mit x_1) die dritte Alphabet-Verteilung der Vorhersage in Rückrichtung genutzt. Durch das Konfidenzmaß werden die Konfidenzen beider Richtungen zusammengefasst und auf einen Wert reduziert. Dies ist jedoch nur eine mögliche Definition des Konfidenzmaßes.

Weitere Konfidenzmaße werden innerhalb dieser Arbeit eingeführt und in Abschnitt 5.5 bezüglich ihrer Kalibrierungsleistung miteinander verglichen. Mithilfe der gleichen Eingabe berechnen die folgenden Konfidenzmaße den aggregierten Konfidenzwert, wobei nicht alle Eingaben im Konfidenzmaß verwendet werden müssen.
Das *AVG* Konfidenzmaß bildet den Durchschnitt der log-Pseudowahrscheinlichkeiten

und verwirft die Eingaben der Rückrichtung. Dahingegen bildet *BIAVG* den Durchschnitt aller log-Pseudowahrscheinlichkeiten. Formal können diese durch

$$AVG(x_1, ..., x_n, y_1, ..., y_n, \hat{y}_1, ..., \hat{y}_n, \hat{\mathbf{y}}_1, ..., \hat{\mathbf{y}}_n) = \frac{\sum y_i}{n} \tag{4.3.2}$$

$$BIAVG(x_1, ..., x_n, y_1, ..., y_n, \hat{y}_1, ..., \hat{y}_n, \hat{\mathbf{y}}_1, ..., \hat{\mathbf{y}}_n) = \frac{\sum y_i + \hat{y}_i}{2n} \tag{4.3.3}$$

definiert werden. Bezogen auf die Pseudowahrscheinlichkeit (nicht log), wird dadurch das Produkt mit einer anschließenden n-ten Wurzel berechnet. Aus dieser Perspektive ergibt sich das *MULT* bzw. *BIMULT* Konfidenzmaß. Dabei werden die log-Pseudowahrscheinlichkeiten aufaddiert, welches dem Produkt der Pseudowahrscheinlichkeiten entspricht. Die beiden Maße können mit

$$MULT(x_1, ..., x_n, y_1, ..., y_n, \hat{y}_1, ..., \hat{y}_n, \hat{\mathbf{y}}_1, ..., \hat{\mathbf{y}}_n) = \sum y_i \tag{4.3.4}$$

$$BIMULT(x_1, ..., x_n, y_1, ..., y_n, \hat{y}_1, ..., \hat{y}_n, \hat{\mathbf{y}}_1, ..., \hat{\mathbf{y}}_n) = \sum y_i + \hat{y}_i \ . \tag{4.3.5}$$

definiert werden. Eine weitere Betrachtung der Konfidenz kann das unsicherste Element sein. Besonders bei 1-Fehler Hypothesen könnte die Konfidenz eines einzelnen fehlerhaften Zeichens deutlich niedriger sein als bei der restlichen Hypothese. Für einen aggregierten Konfidenzwert einer fehlerfreien Hypothese sollte dies eventuell besonders betrachtet werden, wodurch sich das *MIN* bzw. *BIMIN* Konfidenzmaß mit

$$MIN(x_1, ..., x_n, y_1, ..., y_n, \hat{y}_1, ..., \hat{y}_n, \hat{\mathbf{y}}_1, ..., \hat{\mathbf{y}}_n) = \min_i\{y_i\} \tag{4.3.6}$$

$$BIMIN(x_1, ..., x_n, y_1, ..., y_n, \hat{y}_1, ..., \hat{y}_n, \hat{\mathbf{y}}_1, ..., \hat{\mathbf{y}}_n) = \min(\min_i\{y_i\}, \min_i\{\hat{y}_i\}) \tag{4.3.7}$$

ergibt. Der aggregierte Konfidenzwert der gesamten Hypothese entspricht dabei dem Minimum einer bzw. beider Schreibrichtungen.
Die Konfidenzmaße werden in Abschnitt 5.5 weiter untersucht und hinsichtlich des Einflusses auf die Kalibrierung evaluiert. Für die Beam-Search wird in allen Experimenten das *ORI* Konfidenzmaß verwendet, da die Erkennungsleistung durch einen Wechsel leicht abfällt. Das Konfidenzmaß wird also nur nach der Auswahl der besten Hypothese innerhalb der Beam-Search für die Schwellenwert-Funktion verwendet.

Neben den Konfidenzmaßen wird, wie zuvor vorgestellt, Temperature Scaling (vgl. Abschnitt 3.3.1) und LogitNorm (vgl. Abschnitt 3.3.2) verwendet. Für die Optimierung von Temperature Scaling wird ein separater Validierungsdatensatz verwendet. Dieser enthält annotierte Datenpunkte, welche ausschließlich für die Optimierung des Parameters T (vgl. Formel 3.3.1) genutzt werden. Konkret werden die Vorhersagen $\mathbf{Y}$ auf dem Validierungsdatensatz mithilfe dessen Annotationen $\hat{\mathbf{y}}$ und dem Cross-Entropy Loss ϵ_{CE} für die Optimierung genutzt, sodass sich der zu optimierende Fehler zu $loss = \epsilon_{CE}(\mathbf{Y}/T, \hat{\mathbf{y}})$ ergibt. In der Berechnung des Fehlers ist die Formel von Temperature Scaling zu erkennen (vgl. Formel 3.3.1). Der Parameter T soll so angepasst werden, dass der log-Softmax Vektor einer Zeile aus $(\mathbf{Y}/T)_i$ (Verteilung über das Alphabet für ein Zeichen) möglichst dem 1-aus-k Vektor mit einem Maximum bei dem korrekten Zeichen $\hat{\mathbf{y}}_i$ entspricht.
Wie in der Referenzimplementierung der Autoren von Temperature Scaling [GPSW17] in [Ple17], wird der L-BFGS Algorithmus für die Optimierung des Fehlers verwendet. Der Limited-Memory Broyden–Fletcher–Goldfarb–Shanno Algorithmus erzielt bessere Ergebnisse als SGD und konvergiert dabei deutlich schneller, wenn es sich um ein niedrig-dimensionales Optimierungsproblem handelt [LNC+11]. Zur Optimierung wird dabei die Approximation der Hesse-Matrix bzw. dessen Inverse verwendet, um ein Quasi-Newton'sches Update der Gewichte durchzuführen. In der Newton-Raphson-Methode wird die erste sowie zweite Ableitung zur iterativen Bestimmung eines lokalen Minimums verwendet, indem der lokale Suchbereich durch die Taylor-Expansion mit einer Funktion 2. Grades approximiert und minimiert wird.

Als zweite Kalibrierungsmethode wird LogitNorm, wie in Abschnitt 3.3.2 vorgestellt, ohne Veränderungen implementiert. Dafür wird die Loss-Funktion während des Trainings entsprechend Formel 3.3.2 abgeändert. Analog zu [WXC+22], wird der Parameter τ in den Experimenten variiert und die Kalibrierung des Netzwerks untersucht (vgl. Abschnitt 5.5).

Zusammengefasst werden unterschiedliche Konfidenzmaße vor der Filterung des Pseudolabels untersucht und gleichzeitig verschiedene Kalibrierungsmethoden eingesetzt. Insgesamt soll dies das Filtern der Pseudolabel erleichtern und gleichzeitig verbessern.

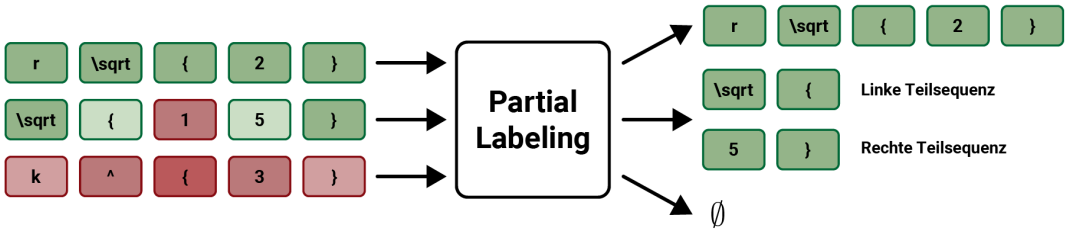


Abbildung 4.3.1: *Vereinfachte Darstellung der Partial Labeling Heuristik, welche die Konfidenzwerte für das vollständige Akzeptieren, Aufteilen oder Ablehnen verwendet. Grün hinterlegte Zeichen sind konfident und rot hinterlegte Zeichen sind unsicher mit einer kleinen Konfidenz.*

4.3.3 *Partial Labeling*

Das Filtern durch einen harten Schwellenwert wird mithilfe des Partial Labelings relaxiert. Statt dem Verwerfen einer Sequenz unterhalb des Schwellenwertes, soll diese mithilfe des Konfidenzwertes der zuvor vorgestellten Konfidenzmaße bei ihrem unsichersten Zeichen zerteilt und für das Training genutzt werden. Da Fehler in den Pseudolabel sich ggf. negativ auf das Training auswirken können, ist eine Reduktion der Fehler in den verwendeten Hypothesen wünschenswert. Gleichzeitig ist nach [RDRS21] die Variation des Datensatzes wichtig, sodass möglichst viele Datenpunkte in das Training aufgenommen werden sollten. Daher ist eine Reduktion des Fehlers bei einer größtmöglichen Variation das Ziel.
Anders als die Literatur zu diesem Thema (z. B. [CST11, NC08]), ist mit dem Partial Labeling nicht die Auswahl einer Hypothese aus einer Menge von potentiellen Hypothesen gemeint. Stattdessen ist damit eine neue Methodik der Arbeit gemeint, die speziell bei Sequenzvorhersagen partiell korrekte Sequenzen für das Training nutzen soll. Die Intuition kommt dabei, ähnlich des *MIN* Konfidenzmaßes, aus der Betrachtung des unsichersten Zeichens einer Sequenz. Sind die Vorhersagen bis zu diesem Zeichen konfident, so soll die Teilsequenz bis zu dem Zeichen für das Training verwendet werden. Da das Netzwerk bi-direktional angelernt wird (vgl. Abschnitt 3.1.1), kann die linke sowie rechte Teilsequenz für das Training genutzt werden. Für die Bestimmung des unsichersten Zeichens können die Konfidenzwerte der Rückrichtung einbezogen werden.
Konkret ist es das Ziel, Hypothesen in einem definierten Konfidenzintervall aufzuteilen, falls eine Entscheidungsregel erfüllt ist. Das grundlegende Vorgehen ist in Abbildung 4.3.1 dargestellt und zeigt die verschiedenen Möglichkeiten bei der Verwen-

dung der Partial Labeling Heuristik. Die Entscheidungsregel prüft, ob das unsicherste Zeichen mehrere Standardabweichungen vom Durchschnitt der restlichen log-Pseudowahrscheinlichkeiten entfernt ist. Das unsicherste Zeichen entspricht dabei dem Minimum des Durchschnitts der log-Pseudowahrscheinlichkeiten beider Richtungen. Formal ergibt sich die Heuristik mit der Entscheidungsregel zu:

$$
\begin{aligned}
&\mathbf{x} \in \mathbb{N}^n, \mathbf{y}, \hat{\mathbf{y}} \in \mathbb{R}^n, \hat{\mathbf{Y}} \in \mathbb{R}^{V \times n} \\
&\qquad c = \mathit{CONF}(\mathbf{x}, \mathbf{y}, \hat{\mathbf{y}}, \hat{\mathbf{Y}}), \omega = \frac{\mathbf{y} + \hat{\mathbf{y}}}{2} \\
&\qquad z = \operatorname*{argmin}_i \omega_i \\
&\qquad \omega' = (\omega_i)_{i \in \{1 \dots n\} \setminus z} \\
&\qquad \mu, \sigma = \mathrm{mean}(\omega'), \mathrm{std}(\omega') \\
&\mathrm{hyp}_{\mathrm{left}} = \begin{cases} \mathbf{x} & \text{, falls } \lambda_{\mathrm{below}} c \geqslant \ln(\tau) \\ \{\mathbf{x}_i\}_{i \in \{1 \dots z-1\}} & \text{, falls } c \geqslant \ln(\tau_p) \wedge \mu - \omega_z \geqslant \lambda_{\mathrm{std}} \sigma e^{c \lambda_{\mathrm{fade}}} \\ \emptyset & \text{, sonst.} \end{cases} \\
&\mathrm{hyp}_{\mathrm{right}} = \begin{cases} \mathbf{x} & \text{, falls } \lambda_{\mathrm{below}} c \geqslant \ln(\tau) \\ \{\mathbf{x}_i\}_{i \in \{z+1 \dots n\}} & \text{, falls } c \geqslant \ln(\tau_p) \wedge \mu - \omega_z \geqslant \lambda_{\mathrm{std}} \sigma e^{c \lambda_{\mathrm{fade}}} \\ \emptyset & \text{, sonst.} \end{cases}
\end{aligned}
\tag{4.3.8}
$$

Die erste Zeile enthält die Eingaben der Heuristik, welche deckungsgleich zu der Eingabe eines Konfidenzmaßes ist (vgl. Abschnitt 4.3.2). Die Eingabe besteht aus den Zeichen der Sequenz $\mathbf{x}$, mit log-Pseudowahrscheinlichkeiten $\mathbf{y}$ bzw. $\hat{\mathbf{y}}$ in Hin- und Rückrichtung, sowie den Alphabet-Verteilungen der Rückrichtung $\hat{\mathbf{Y}}$. In der zweiten Zeile wird der aggregierte Konfidenzwert c mithilfe des konfigurierbaren Konfidenzmaßes aus Abschnitt 4.3.2 gebildet. Darüber hinaus wird der bi-direktionale Durchschnitt der log-Pseudowahrscheinlichkeiten in ω abgespeichert. Dann wird mit z das Minimum aus ω bestimmt. In ω' sind alle durchschnittlichen log-Pseudowahrscheinlichkeiten aus ω enthalten, außer dem zuvor bestimmten Minimum. Danach wird mit μ bzw. σ der Mittelwert bzw. die Standardabweichung der übrigen log-Pseudowahrscheinlichkeiten berechnet.

Schließlich folgt die eigentliche Heuristik, welche die linke bzw. rechte Teilsequenz bestimmt. Mit dem ersten Hyperparameter $\lambda_{\mathrm{below}} \in \{0, 1\}$ kann der Anfang des Intervalls gesteuert werden, in welchem die Heuristik angewandt wird. Gilt $\lambda_{\mathrm{below}} = 1$,

wird die Heuristik auf alle Sequenzen mit Konfidenz $c \in [\tau_p, \tau)$ angewandt. Dabei ist τ der harte Schwellenwert der vorherigen Self-Training Methode. Oberhalb des Schwellenwertes wird die Vorhersage vollständig in das Training aufgenommen. Dies entspricht also dem üblichen Filtern des zuvor beschriebenen Self-Trainings (vgl. Abschnitt 3.2.1). Bei $\lambda_{\text{below}} = 0$ ergibt sich das Intervall zu $c \in [\tau_p, 1]$. Dadurch wird τ ignoriert, sodass eine Aufteilung der Sequenz immer geprüft wird. Durch die Variable kann daher entschieden werden, ob sichere Sequenzen vollständig in das Training aufgenommen werden oder ob diese mit der Heuristik aufgeteilt werden. Mit dem weiteren Schwellenwert τ_p kann eine Untergrenze zum Filtern von sehr unsicheren Sequenzen definiert werden.

Mit der Entscheidungsregel $\mu - \omega_z \geqslant \lambda_{\text{std}} \sigma e^{c\lambda_{\text{fade}}}$ wird der Abstand $\mu - \omega_z$ des unsichersten Zeichens zum restlichen Durchschnitt mit der Standardabweichung der übrigen log-Pseudowahrscheinlichkeiten verglichen. Ist der Abstand größer, so werden die Teilsequenzen als Pseudolabel verwendet. Intuitiv sollen damit Ausreißer gefunden werden, die einen vordefinierten Mindestabstand zur restlichen Verteilung aufweisen. Die Standardabweichung wird jedoch mit zwei weiteren Hyperparametern modifiziert. Mit λ_{std} kann eine simple Anzahl an Standardabweichungen festgelegt werden, welche überschritten werden muss, damit ein unsicheres Zeichen als Ausreißer gilt. Durch $e^{c\lambda_{\text{fade}}}$ kann die Anzahl an benötigten Standardabweichungen reduziert werden. Die Idee dabei ist, dass der benötigte Abstand mit genereller Unsicherheit der gesamten Hypothese sinkt. Anders ausgedrückt, je unsicherer eine Hypothese, desto einfacher ist es diese aufzuteilen statt abzuweisen. Durch $c\lambda_{\text{fade}}$ wird die Pseudowahrscheinlichkeit mit λ_{fade} potenziert, da c eine log-Pseudowahrscheinlichkeit ist. Mithilfe der e-Funktion wird dies zurück in das Intervall $[0, 1]$ abgebildet. Je größer λ_{fade} ist, desto weniger stark müssen Ausreißer vom Mittelwert der übrigen Zeichen abweichen.
Für die Experimente werden drei verschiedene Kombinationen an Hyperparametern verwendet. Dabei soll der Einfluss von vielen bzw. wenigen Teilsequenzen im Training untersucht werden. Die Hyperparameter für die drei „Profile“ ergeben sich zu:

$$\begin{aligned} \textit{High} &= \left\{ \lambda_{\text{below}} = 1, \tau_p = 0, \lambda_{\text{std}} = 0, \lambda_{\text{fade}} = 0 \right. \\ \textit{Med} &= \left\{ \lambda_{\text{below}} = 1, \tau_p = 0.05, \lambda_{\text{std}} = 3.5, \lambda_{\text{fade}} = 2.0 \right. \\ \textit{Low} &= \left\{ \lambda_{\text{below}} = 1, \tau_p = 0.15, \lambda_{\text{std}} = 5.5, \lambda_{\text{fade}} = 1.0 \right. \end{aligned} \tag{4.3.9}$$

Im „*High*“ Profil werden alle Hypothesen unterhalb des harten Schwellenwertes τ anhand des unsichersten Zeichens aufgeteilt. Das „*Med*“ Profil verwirft Hypothesen unterhalb einer Konfidenz von 0.05. Zudem ist mit $\lambda_{\text{std}} = 3.5$ ein Mindestabstand

definiert, der jedoch quadratisch mit der Unsicherheit der Hypothese abfällt. In diesem Profil sollen weniger Hypothesen mit der Partial Labeling Heuristik aufgeteilt werden als mit den Hyperparametern des „*High*“ Profils. Die Hyperparameter im „*Low*“ Profil sind dagegen noch strikter, sodass ein größerer Abstand zum Mittelwert vorliegen muss und dieser gleichzeitig weniger stark mit der generellen Unsicherheit sinkt. Dadurch soll die Menge an aufgeteilten Hypothesen weiter sinken, sodass generell unsichere Sequenzen verworfen werden, während deutliche Ausreißer zum Aufteilen führen. Die verschiedenen Profile werden in Abschnitt 5.6.2 miteinander verglichen.

5 EXPERIMENTE UND EVALUATION

Die vorgestellten Methoden werden in den folgenden Experimenten schrittweise kombiniert und untersucht. Die Experimente bauen dabei auf dem vollständig-überwacht angelernten CoMER Modell zur handschriftlichen mathematischen Formelerkennung auf. Das Ziel ist es, eine semi-überwachte Lernmethodik inkrementell auf einem Datensatz zu entwickeln und danach auf einem weiteren Datensatz zu validieren. Folgende Forschungsfragen sollen dabei beantwortet werden: In welchem Maß können nicht-annotierte Daten die Erkennungsleistung eines Neuronalen Netzes zur handschriftlichen mathematischen Formelerkennung verbessern und inwieweit kann der annotierte Datenaufwand reduziert werden.

Zunächst wird das CoMER Modell vollständig-überwacht mit dem gesamten Datensatz angelernt. Anschließend werden nur Teile des Datensatzes für das Training verwendet. Dadurch soll der Vergleich zwischen einem vollständig-überwacht und semi-überwacht angelernten System möglich werden, da die nicht-annotierten Datenpunkte als zusätzliche Daten in den semi-überwachten Lernmethoden verwendet werden. Die vorherige Auswahl an annotierten Datenpunkten bleibt währenddessen unverändert. Damit soll der Einfluss der jeweiligen Lernmethodik unter Verwendung der nicht-annotierten Daten evaluierbar werden.

Für das semi-überwachte Anlernen wird zunächst das Self-Training ohne weiterführende Veränderungen zur Verbesserung der vollständig-überwacht angelernten Netze angewendet (vgl. Abschnitt 4.3.1). Um eine Art theoretisches Maximum der grundlegenden Lernmethodik zu bestimmen, wird daraufhin ein perfektes Konfidenzmaß verwendet, welches das Wissen über die Annotation der nicht-annotierten Daten ausnutzt. Dadurch kann der erlaubte Fehler und die Varianz der Schwellenwert-passierenden Sequenzen gesteuert werden. Eine erhöhte Varianz wird durch zufällige Auswahl eines nicht-annotierten Datenpunktes mit gleicher Fehleranzahl erreicht.

Anschließend wird die Kalibrierung des Netzwerks untersucht und als mögliche Verbesserung für die Erkennungsleistung identifiziert. Dabei werden die verschiedenen Konfidenzmaße, sowie die Kalibrierungsmethoden Temperature Scaling bzw. LogitNorm gegenübergestellt und evaluiert (vgl. Abschnitt 4.3.2). Die daraus gewon-

nen Kenntnisse werden für eine weitere Self-Training Versuchsreihe verwendet. Dabei wird das Self-Training mit den Kalibrierungsmethoden erweitert und der Zugewinn der Erkennungsleistung erneut evaluiert.

Daraufhin werden verschiedene Erweiterungen der Methodik betrachtet. Einerseits werden synthetische Daten in einem Vortraining verwendet (vgl. Abschnitt 4.1.1). Nach dem Vortraining werden erneut nur Teile des Datensatzes für ein vollständig-überwachtes Anlernen genutzt, wonach das kalibrierte Self-Training folgt. Andererseits wird der harte Schwellenwert im Self-Training mit der Partial Labeling Heuristik (vgl. Abschnitt 4.3.3) ersetzt und der Einfluss auf die Erkennungsrate untersucht.

Die letzte Erweiterung verwendet die Testdaten als nicht-annotierte Trainingsdaten. Das Ziel ist eine Anpassung auf die konkrete Zieldomäne im Produktionsbetrieb des Netzwerks. In den Experimenten werden die Testdaten als Daten der Zieldomäne verwendet, sodass diese im Self-Training zur Verbesserung der Erkennungsleistung genutzt werden.

Schließlich werden einzelne Experimente der Zwischenschritte auf einem zweiten Datensatz wiederholt, um die grundlegenden Erkenntnisse auf diesem zu validieren. Die Ergebnisse der Experimente werden dabei mit den Ergebnissen der Literatur verglichen. Dabei werden jedoch nur Arbeiten mit vollständig-überwachten Ansätzen herangezogen, da zum Zeitpunkt dieser Arbeit keine weiteren Arbeiten zum semi-überwachten Lernen im Bereich der handschriftlichen mathematischen Formelerkennung bekannt sind.

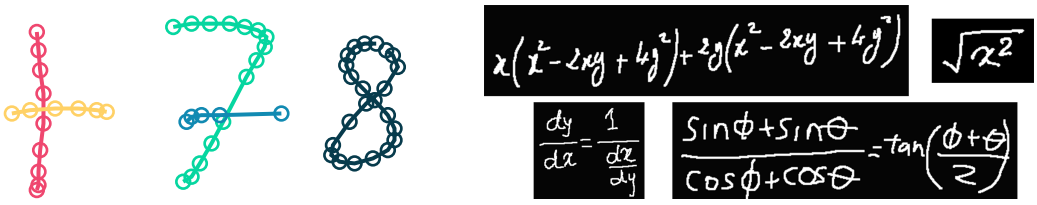
5.1 DATENSÄTZE

Für die Experimente werden insgesamt drei Datensätze verwendet: Ein synthetisch generierter und zwei mit handgeschriebenen mathematischen Formeln. Auf einem handgeschriebenen Datensatz wird die Lernmethodik schrittweise aufgebaut und schließlich auf dem zweiten Datensatz validiert. Der synthetisch erzeugte Datensatz wird für ein Vortraining verwendet. Dabei soll die Syntax von allgemein wohlgeformten LaTeX Sequenzen erlernt werden, sodass dadurch ein implizites Sprachmodell mit erlernt wird [SRN17].

5.1.1 *CROHME*

Für das schrittweise Aufbauen sowie Evaluieren der Lernmethodik wird der CROHME Datensatz genutzt. Der Datensatz liegt in verschiedenen Variationen vor, da diese im Rahmen der Wettbewerbe „**C**ompetition on **R**ecognition of **O**nline **H**andwritten **M**athematical **E**xpressions“ in verschiedenen Jahren veröffentlicht und verwendet wurden [MVGZG14, MVGZG16, MZM+19]. Der Datensatz wird in fast jeder Publikation im Bereich der handschriftlichen Formelerkennung für den Vergleich der Erkennungsraten verwendet. In dieser Arbeit werden die Testdatensätze der Jahre 2014, 2016 sowie 2019 genutzt. Für das Training werden die Trainingsdaten aus dem Wettbewerb im Jahr 2019 genutzt [MZM+19], welche aus 8 836 annotierten Datenpunkten bestehen.

Für Online-Erkennungsaufgaben liegen die Schreibtrajektorien der handschriftlich geschriebenen mathematischen Formeln vor. Die Abtastpunkte der Trajektorie sind dabei in Segmente aufgeteilt, die einen kontinuierlichen Strich des Nutzers darstellen. Durch Verbinden der Abtastpunkte mit einer Linie mit einer vordefinierten Dicke können die Schreibtrajektorien zu einem Bild konvertiert werden. Die daraus entstehenden Bilder können für Offline-Erkennungsaufgaben genutzt werden. In Abbildung 5.1.1a sind die Punkte der Schreibtrajektorie einer handgeschriebenen Formel des CROHME Datensatzes dargestellt, wobei die Segmente durch die unterschiedliche Färbung zu erkennen sind. Zusätzlich ist in Abbildung 5.1.1b das schwarz-weiß Bild der Formeln abgebildet, welches mit der beschriebenen Methode gezeichnet wurde. Die hier verwendeten Datensätze bestehen aus schwarz-weiß Bildern von insgesamt rund 10 000 Formeln und den dazugehörigen Annotationen in Form von Sequenzen von LaTeX Makros.



(a) (b)

Abbildung 5.1.1: *(a) Schreibtrajektorie einer Formel in welcher die einzelnen kontinuierlichen Segmente farblich hervorgehoben und die einzelnen Abtastpunkte durch Kreise dargestellt sind. (b) Beispiele aus dem konvertierten CROHME Offline-Datensatz.*

Neben Buchstaben und Zahlen beinhalten die Formeln unter Anderem auch griechische Buchstaben, Summenzeichen, Allquantoren, Brüche, Hoch- oder Tiefstellung. Wurzelzeichen oder Brüche können dabei auch verschachtelt auftreten. Meist handelt es sich bei einer Formel um einen simplen Ausdruck, jedoch sind auch Gleichungen und Ungleichung enthalten. Die Datensätze wurden dabei nach festen Regeln erstellt. Ausgehend von Formeln aus Wikipedia oder kleineren privaten Datensätzen von verschiedenen Universitäten für 2014 und 2016 bzw. arXiv-Veröffentlichungen für 2019, wurde auf ein annäherndes Gleichgewicht der Symbolverteilungen im Trainings- sowie Testdatensatz geachtet [MVGZG14, MZM+19]. Zusätzlich wurde die Komplexität der Formeln berücksichtigt, welche z. B. durch die Tiefe des MathML Baumes oder die Anzahl an Wechsel der Tief- bzw. Hochstellung gemessen werden kann. Durch die analytisch optimierte Zusammenstellung des Datensatzes sollen die beiden Teilmengen disjunkt sein, aber dennoch die gleiche Problemdomäne in einer ähnlichen Komplexität widerspiegeln.

5.1.2 *NTCIR-12 MathIR*

Ebenfalls im visuellen Format von CROHME sind die Formeln des synthetisch generierten NTCIR-12 MathIR Datensatzes. NTCIR-12 MathIR enthält mathematische Formeln verschiedener Quellen im MathML sowie LaTeX Format. In dieser Arbeit werden die Formeln aus dem Wikipedia Corpus verwendet. Der Corpus enthält 592 443 Formeln aus 319 689 Wikipedia Artikeln [ZAK+16]. Da die Formeln bereits als LaTeX Sequenz vorliegen, müssen diese in einem Vorverarbeitungsschritt gerendert werden, um als Eingabe für das Image-To-Sequence Problem zu dienen. Dafür werden die einzelnen Zeichen mithilfe von Glyphen dargestellt. Die Glyphen können dabei aus einer Schriftart stammen oder aus einem Datensatz handgeschriebener Zeichen. Der hier verwendete Datensatz nutzt die Standard-Schriftart von LaTeX für Formeln „Latin Modern Math“ [Knu12]. Mithilfe der Schriftart werden die Formeln im visuellen Format des CROHME Datensatzes gerendert, wodurch eine weiße Schrift auf einem schwarzen Hintergrund gezeichnet wird.
Zuvor werden die Formeln jedoch mithilfe des verwendeten Alphabets gefiltert, damit nur valide Zeichen in den Formeln vorkommen. Insgesamt besteht der Datensatz aus

$$-0.000739 - (235 \times 10^{-12}) \times N^2$$

Abbildung 5.1.2: *Beispiel einer synthetisch generierten Formel, basierend auf dem NTCIR-12 Math IR Datensatz [ZAK+16].*

97 547 synthetisch erzeugten Formeln. Ein Beispiel von einer solchen Formel ist in Abbildung 5.1.2 dargestellt. Für das Training werden 87 793 und für die Validierung 9 754 Formeln verwendet. Der Datensatz wurde dabei im Rahmen der Bachelorarbeit „Datensynthese für die Erkennung mathematischer Ausdrücke mit tiefen neuronalen Netzen“ von M. Frichert an der TU Dortmund erstellt [Fri23].

5.1.3 *HME100K*

Der zweite und zur Validierung der Methodik genutzte Datensatz handschriftlicher mathematischer Formeln besteht aus Scans bzw. Fotos. Im Gegensatz zu CROHME wird also keine Trajektorie für die Erstellung eines schwarz-weiß Bildes verwendet. Einige Beispiele des HME100K Datensatzes sind in Abbildung 5.1.3 dargestellt und zeigen den deutlichen Unterschied zu den Datenpunkten des CROHME Datensatzes. Der HME100K Datensatz beinhaltet 99 106 Bilder, aufgeteilt in 74 502 Datenpunkte für das Training und 24 607 für das Testen des Modells [YLD+22].
Nach [YLD+22] wurden mithilfe einer Internetanwendung mehr als eine Milliarde Formeln gesammelt, welche mithilfe von Vorhersagen eines vortrainierten Netzes gefiltert und von Duplikaten befreit worden sind. Aus den verbleibenden 700 000 Formeln wurde der Datensatz in drei Schritten erstellt. Zunächst wurde ein bereits trainiertes DenseWAP Modell [ZDD18] genutzt, um die Sequenzen der 700 000 Bildern zu bestimmen, welche auf einer öffentlichen Plattform händisch korrigiert wurden. Im zweiten Schritt wurden 600 000 der korrigierten Vorhersagen zufällig ausgewählt und für ein neues Training des DenseWAP Modells verwendet. Im dritten Schritt wurden erneut Vorhersagen für die restlichen 100 000 Bilder generiert. Falls die Vorhersagen nicht mit den korrigierten Sequenzen aus dem ersten Schritt übereinstimmen, wurden diese erneut auf der Plattform händisch korrigiert. Zusätzlich wurden einige Datenpunkte entfernt, da diese keine Formeln enthielten, sodass die etwas unter 100 000 verbleibenden Datenpunkte (99 106) den Datensatz bilden [YLD+22]. In einem Drei zu Eins Verhältnis werden diese für das Training und Testen aufgeteilt. Die Zusam-

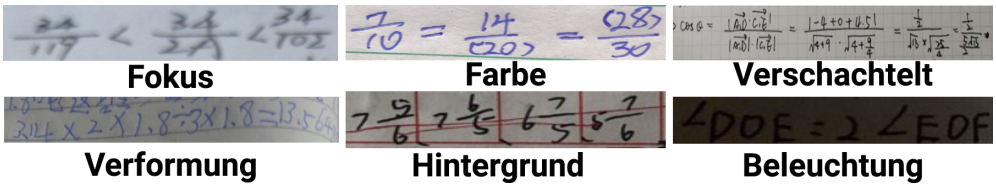


Abbildung 5.1.3: *Beispiele verschiedener Formeln aus dem HME100K Datensatz [YLD+22], welche die verschiedenen Imperfektionen zeigen.*

menstellung des Trainings- bzw. Testdatensatzes wurde zufällig durchgeführt, statt, wie in CROHME, hinsichtlich Komplexitätsmetriken und Gleichgewichten optimiert zu werden. Zusätzlich werden die Testdaten in drei disjunkte Untermengen „Easy“, „Medium“ und „Hard“ aufgeteilt. Dafür wird die Länge und die Komplexität der Formel betrachtet [YLD+22].

Da es sich um Scans bzw. Fotos von handschriftlichen mathematischen Formeln handelt, enthalten die Datenpunkte verschiedene Imperfektionen. Dadurch sind die Datenpunkte, im Vergleich zum CROHME Datensatz, auf der visuellen Ebene deutlich komplexer. Zum Einen enthalten die Datenpunkte komplexere und variierende Hintergründe. Beispielsweise sind karierte Hintergründe mit verschmierten Linien oder ein leichter Farbverlauf in dem Datensatz vorhanden. Zum Anderen kommen nicht optimale Bedingungen beim Fotografieren der Formeln vor, wie z. B. ein falscher Fokus, eine kontrastarme Beleuchtung oder eine Verformung des Blattes (vgl. Abbildung 5.1.3). Dadurch, dass die Datenpunkte nicht mithilfe einer Schreibtrajektorie gerendert werden, sondern durch Scannen und Fotografieren entstanden sind, sollen diese die Realität besser widerspiegeln [YLD+22]. Nach [YLD+22] ist die Vielzahl an Schreibern und Schreibstilen ein Alleinstellungsmerkmal des Datensatzes. Darüber hinaus ist sowohl die maximale als auch die durchschnittliche Sequenzlänge größer als beim CROHME Datensatz. Beim genaueren Betrachten der Datenpunkte wird jedoch ersichtlich, dass die Komplexität der Formeln und des Alphabets im Vergleich zu CROHME abgenommen hat. Zwar sind viele verschachtelte Brüche enthalten, jedoch werden z. B. keine Betragsstriche in den Formeln verwendet. Zudem enthalten 11.14% statt 20.44% (CROHME) der Formeln mindestens ein griechisches Zeichen. Darüber hinaus enthalten 27.15% statt 53.31% der Formeln mindestens eine Hoch- oder Tiefstellung. Insgesamt deuten sowohl diese Zahlen, als auch die qualitative Betrachtung mehrerer Beispiele des HME100K Datensatz darauf hin, dass die Formeln von Schülern geschrieben wurden. Viele der Datenpunkte enthalten z. B. Ausschnitte anderer Formeln, mit Bleistift oder rot geschriebene Korrekturhaken, eingekreiste Zahlen oder chinesische Zeichen.

5.2 EVALUATIONSMETRIKEN

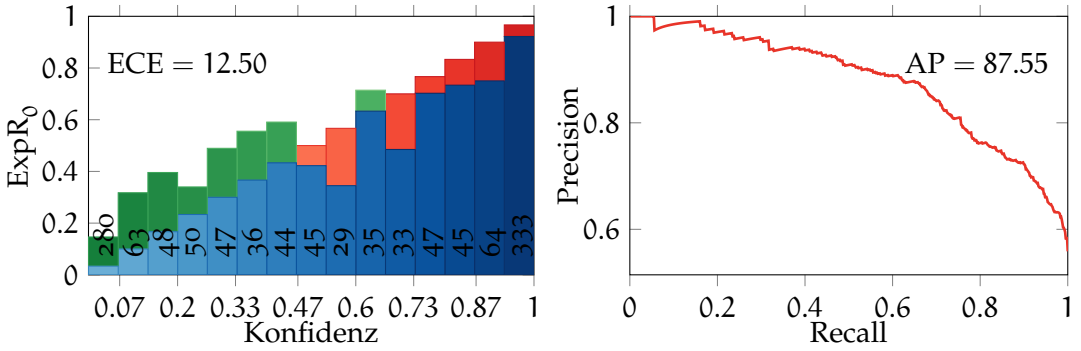
Um die Erkennungsleistung auf den verschiedenen Datensätzen zu messen und die verschiedenen Experimente miteinander zu vergleichen, werden Metriken benötigt, welche die Erkennungsleistung widerspiegeln.

Dafür wird die 0-Fehler Erkennungsrate ($ExpR_0$) verwendet, die mithilfe des LgEval Tools des CROHME Wettbewerbs bestimmt wird [MZM+19]. Die 0-Fehler Metrik bestimmt dabei den Anteil an Sequenzen, die vollständig fehlerfrei erkannt werden. Dazu wird die vorhergesagte LaTeX Sequenz mithilfe von Pandoc in einen MathML Baum umgeformt und anschließend mithilfe von LgEval zu einem *Symbolic Label Graph* konvertiert. Durch den Graph ist es möglich zwischen symbolischen, relationalen oder strukturellen Fehlern zu differenzieren. Da eine mathematische Formel durch mehrere verschiedene LaTeX Sequenzen erzeugt werden kann, können dadurch „falsche" Fehler in der Metrik vermieden werden. Als Beispiel dafür sei die Reihenfolge der Hoch- bzw. Tiefstellung genannt, die in der vorhergesagten Sequenz vertauscht werden kann, ohne die resultierende Formel zu verändern.
Zur Bestimmung der $ExpR_0$ auf den HME100K Daten wird jedoch nur der direkte Vergleich der Vorhersage und der Annotation genutzt. Da in HME100K LaTeX Makros verwendet werden, die nicht mit Pandoc kompatibel sind, können diese nicht zu einem LgEval-kompatiblen MathML Baum überführt werden. Dadurch kann die „wahre" $ExpR_0$ bei diesen Experimenten noch höher sein als angegeben.

Neben der Erkennungsleistung wird auch die Kalibrierung der Netzwerke untersucht. Die Kalibrierung kann z. B. mithilfe des *Expected* bzw. *Maximum Calibration Error* (ECE und MCE) [NCH15] quantifiziert werden. Intuitiv beschreibt der ECE die Abweichung vom erwarteten Fehler auf Grundlage der Konfidenz. Werden 100 Hypothesen mit einer Konfidenz von 0.8 vorhergesagt, sollten bei einer perfekten Kalibrierung genau 80 vollständig korrekt sein. Der Erwartungswert der Abweichung tatsächlich korrekter Hypothesen von dem Durchschnitt in einem beliebigen Konfidenzintervall wird im ECE erfasst. Dahingegen misst der MCE die maximale Abweichung von dem erwarteten Fehler. In dieser Arbeit wird, analog zur Literatur, vorrangig die ECE Metrik verwendet [Wal69, DKI+21, RDRS21, NCH15, GPSW17]. Formal kann der ECE durch eine Aufteilung des Konfidenzraumes in K disjunkte Intervalle (Bins) und mit

$$\hat{ECE} = \sum_{i=1}^{K} P(i) \cdot |ExpR_0(i) - \mu_i| \tag{5.2.1}$$

annähernd bestimmt werden [NCH15]. Dabei ist $P(i)$ die Gewichtung des einzelnen Bins, gegeben durch n_i/N bei n_i Datenpunkten im Bin i und einer Gesamtanzahl N. Durch $|ExpR_0(i) - \mu_i|$ wird der Kalibrierungsfehler eines einzelnen Bins bestimmt. Dieser beschreibt die Abweichung der tatsächlichen 0-Fehler Erkennungsrate $ExpR_0(i)$ von der durchschnittlichen Konfidenz μ_i im Bin i. Die Abweichung in den einzelnen Bins kann auch graphisch dargestellt werden. Wie in Abbildung 5.2.1a zu sehen,



(a) **(b)**

Abbildung 5.2.1: *(a) ECE-Diagramm eines mit 35% der CROHME$_{19}$ Trainingsdaten angelernten Netzwerks, welches das BIMIN Konfidenzmaß nutzt. (b) Die dazugehörige PR-Kurve.*

entspricht die von der Diagonale abweichende Fläche dem Kalibrierungsfehler. Ein Bin oberhalb der Diagonale bedeutet eine *Underconfidence* des Netzwerks, da die tatsächliche ExpR$_0$ größer ist als der Mittelwert der Konfidenz des Bins erwarten lässt. Diese Fläche ist in Abbildung 5.2.1a grün eingezeichnet, wobei die blaue Fläche der erwarteten ExpR$_0$ des Bins entspricht. Ist die ExpR$_0$ niedriger als der erwartete Mittelwert des Bins, so handelt es sich um *Overconfidence*. Die blaue Fläche entspricht dann der tatsächlichen ExpR$_0$ und der Abstand zur erwarteten ExpR$_0$ ist rot dargestellt.

Eine weitere Metrik, welche bei der Kalibrierung beachtet werden sollte, ist die *Average Precision* (AP). Während des Self-Trainings werden Sequenzen aufgrund des Konfidenzwertes und des harten Schwellenwertes ausgewählt. Der Schwellenwert teilt den Konfidenzraum in zwei Intervalle. Dabei ist die Sortierung der Sequenzen im Konfidenzraum relevant, da damit die Menge an Schwellenwert-passierenden Sequenzen maßgeblich beeinflusst wird. Daher ist die Sortierung eines Konfidenzmaßes relevant für die Auswahl von möglichst fehlerfreien Sequenzen, welche mit der AP gemessen werden kann. Eine hinsichtlich der AP optimale Sortierung würde zunächst alle korrekten Sequenzen zurückgeben, gefolgt von allen falschen. Die AP entspricht der Fläche unter der Precision-Recall Kurve (PR-Kurve) und kann durch Betrachtung der nach Konfidenz sortierten Vorhersageliste berechnet werden [Pow11]. Ist die Vorhersageliste absteigend nach Konfidenz sortiert, kann bei jedem Datenpunkt der Recall und die Precision berechnet werden. Die Precision entspricht dem Anteil korrekter Vorhersagen bis einschließlich dem i-ten Datenpunkt. Der Recall beschreibt

welcher Anteil aller korrekten Hypothesen bis zu diesem Datenpunkt enthalten sind. Wird der Recall nun auf der X-Achse und die Precision auf der Y-Achse abgetragen, ergibt sich die PR-Kurve. Existieren für einen Recall-Wert mehrere Precision-Werte, wird das Maximum genutzt. Die Fläche unterhalb der PR-Kurve entspricht dann der AP [Pow11]. Die AP liegt im Intervall $[0, 1]$ mit einem optimalen Wert von 1. Im Allgemeinen wird für die Berechnung der Precision und des Recalls die Anzahl an *True Positives*, *False Positives*, *True Negatives* bzw. *False Negatives* verwendet. Die Begriffe lassen sich anhand der binären Klassifikation erklären: Der Klassifikator soll „True" oder „False" für einen Datenpunkt ausgeben. Ein True Positive beschreibt die korrekte Klassifikation eines „True" Datenpunktes. Wird dieser fälschlicherweise als „False" klassifiziert, handelt es sich um ein False Negative. Analog ergeben sich die anderen beiden Begriffe bei der korrekten Klassifikation eines „False" Datenpunktes (True Negative) und bei der falschen Klassifikation (False Positive). Die Anzahl an Vorhersagen in den verschiedenen Kategorien wird zur Berechnung der Precision und des Recalls in

$$\begin{aligned} \text{Recall}_i &= \frac{\text{TP}_i}{\text{TP} + \text{FN}} \\ \text{Precision}_i &= \frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i} \end{aligned} \tag{5.2.2}$$

verwendet [Pow11]. Dabei entspricht TP_i (analog für FP) der Anzahl an True Positives in der sortierten Vorhersageliste bis einschließlich des i-ten Datenpunktes. Zum Erstellen der PR-Kurve werden die $(\text{Recall}_i, \text{Precision}_i)$ Paare nach Recall aufsteigend sortiert. Das erste Vorkommen eines größeren Recalls mit der entsprechenden Precision_i wird dann im Graph abgetragen. Wird die entstehende Kurve integriert, ergibt sich die AP. Ein Beispiel einer PR-Kurve mit der AP ist in Abbildung 5.2.1b dargestellt. Für die Berechnung der Precision und des Recalls werden alle korrekten Sequenzen als True Positives angesehen. Ist eine Vorhersage korrekt, wird diese in die Zwischensumme TP_i aufgenommen. Ist diese jedoch fehlerbehaftet, wird sie zu FP_i hinzugefügt. TP_i entspricht also allen korrekten und FP_i allen fehlerbehafteten Vorhersagen bis zu dem i-ten Datenpunkt in der sortierten Liste. Da es keine „Negative" Klassifizierung gibt, entfällt FN. Mit dieser Definition kann die Veränderung der Sortierung durch ein anderes Konfidenzmaß sowie durch eine andere Kalibrierungsmethode verglichen werden. Werden stattdessen alle Sequenzen als True Positives angesehen, kann die Erkennungsleistung über alle Datenpunkte in der PR-Kurve abgelesen werden. Der maximale Recall entspricht dann der ExpR_0 über alle Datenpunkte. Dadurch kann die Sortierung und die Erkennungsleistung gleichzeitig miteinander verglichen werden. Dies kann z. B. beim Überlagern von zwei PR-Kurven im Falle des Partial Labelings

nützlich sein. Wird die Heuristik ab einem definierten Schwellenwert angewandt, so kann die $ExpR_0$ ab diesem Punkt durch fehlerfreie Teilsequenzen steigen. Dadurch verändert sich die Anzahl an korrekten (Teil-)Vorhersagen, wobei die Gesamtmenge an Vorhersagen gleich bleibt. Da dies aber die Skalierung verändert, sollte die Wahl von TP gleich bleiben, wenn zwei Kurven miteinander verglichen werden (vgl. Abbildung A.1.4).

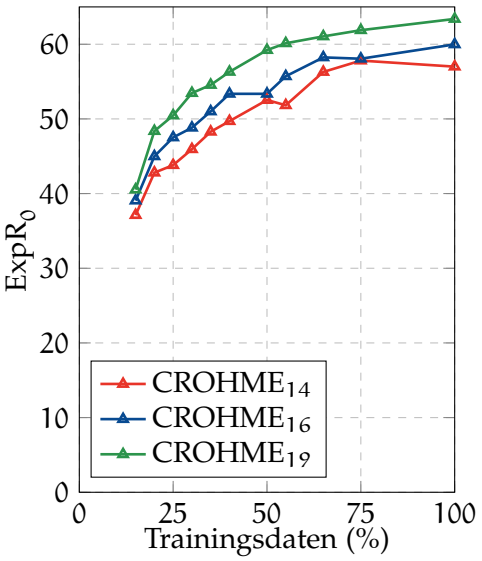
5.3 FULLY-SUPERVISED LEARNING

Innerhalb des ersten Experiments soll die Implementierung und das Training des CoMER Modells getestet werden. Das Ziel ist es dabei, die in [ZG22a] veröffentlichen Zahlen zu reproduzieren. Dafür wird der gesamte $CROHME_{19}$ Trainingsdatensatz für das vollständig-überwachte Anlernen des Netzwerks verwendet. Das Vorgehen und die verwendeten Hyperparameter sind dabei in Abschnitt 4.1 beschrieben. Anschließend werden nur Teile der Trainingsdaten verwendet, um die erreichte Erkennungsleistung mit den späteren semi-überwachten Experimenten zu vergleichen. Die genutzten Trainingsdaten werden dabei zufällig aus allen annotierten Datenpunkten ausgewählt. Mithilfe des *Seeds* des Generators für Zufallszahlen kann sichergestellt werden, dass immer der gleiche Teil der Trainingsdaten ausgewählt wird.

Die Ergebnisse des vollständig-überwachten Trainings sind in Tabelle 5.3.1a dargestellt und werden zusätzlich mit einigen derzeitigen State-of-the-Art Modellen verglichen. Die Erkennungsleistung aus [ZG22a] kann reproduziert werden und ist im Falle der $CROHME_{16}$ bzw. $CROHME_{19}$ Testdaten sogar leicht über den veröffentlichten Zahlen. Die derzeitige State-of-the-Art Erkennungsleistung erreicht dabei das RNN-basierte CAN-ABM Modell aus [LYL+22]. Kurz dahinter folgt das Transformer-basierte CoMER Modell [ZG22a]. Da das CoMER Modell nur den Coverage Mechanismus zum BTTR Modell [ZGY+21] hinzufügt, kann der Einfluss des Korrekturterms auf die Erkennungsleistung evaluiert werden. Durch den Korrekturterm wird die Erkennungsleistung zwischen drei und vier Prozentpunkten verbessert. Zusätzlich kann der Einfluss der ScaleAug auf die Erkennungsleistung verglichen werden. Das BTTR Modell wird in [ZGY+21] ohne und in [ZG22a] mit ScaleAug trainiert. Da das restliche Modell und die Methodik gleich bleibt, wird der Zugewinn an Erkennungsleistung zwischen 1.2 und 6.6 Prozentpunkten durch die Augmentierung erreicht.

Modell (% Train_{19})	CROHME		
	14	16	19
BTTR [ZGY+21]	54.0	52.3	53.0
CAN-ABM* [LYL+22]	65.9	63.1	64.5
BTTR* [ZG22a]	55.2	56.6	59.6
CoMER* [ZG22a]	59.3	59.8	63.0
CoMER* (100%)	57.0	60.0	63.4
CoMER* (75%)	57.8	58.1	61.9
CoMER* (65%)	56.3	58.2	61.1
CoMER* (50%)	52.5	53.4	59.2
CoMER* (35%)	48.3	51.0	54.5
CoMER* (25%)	43.8	47.5	50.5
CoMER* (15%)	37.1	39.1	40.5

(a)



(b)

Tabelle 5.3.1 & Abbildung 5.3.1: *(a) ExpR_0 (%) von vollständig-überwacht angelernten Modellen im Vergleich mit der Literatur auf den CROHME Testdatensätzen. Ansätze mit * nutzen Online-Augmentierungen. (b) ExpR_0 bei Variation der Menge an annotierten Trainingsdaten.*

Wenn die Netzwerke mit weniger Trainingsdaten vollständig-überwacht trainiert werden, sinkt die Erkennungsleistung. Wie in Abbildung 5.3.1b zu sehen, ist dies jedoch kein linearer Zusammenhang. Zwischen 15% und 50% Trainingsdaten steigt bzw. fällt die Erkennungsrate deutlich schneller als in den restlichen 50%. Mehr Trainingsdaten beeinflussen also ab einem gewissen Punkt nur noch marginal die Erkennungsrate. Damit die Modellparameter von CoMER überhaupt konvergieren, müssen mindestens rund 15% (1325 Datenpunkte) der Trainingsdaten verwendet werden. Falls 10% oder weniger Trainingsdaten genutzt werden, sinkt die ExpR_0 auf weniger als 0.1%. Eine mögliche Ursache dafür kann die gewählte Lernrate zu Beginn des Trainings sein oder das Verhältnis von anlernbaren Modellparametern zu annotierten Datenpunkten.

Das Ziel in den folgenden Experimenten ist es die restlichen Trainingsdaten als nicht-annotierte Daten im semi-überwachten Training zu nutzen. Dadurch soll der Abstand zu einem Netzwerk mit Zugriff auf alle Annotationen verringert werden. So

soll beispielsweise die Erkennungsleistung des mit 15% Trainingsdaten angelernten Netzwerks verbessert werden, indem die 85% nicht-annotierten Datenpunkte im Self-Training verwendet werden.

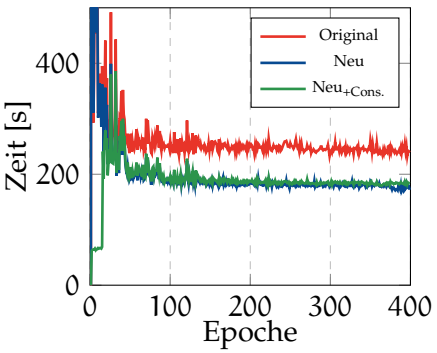
5.3.1 *Inferenzgeschwindigkeit mit Pruning-Methoden*

Während des Self-Trainings müssen Vorhersagen für alle nicht-annotierten Datenpunkte mithilfe der Inferenz generiert werden. Da dies in jeder Epoche durchgeführt wird, muss die Inferenzdauer für die praktische Umsetzung so gering wie möglich gehalten werden. Da aber bereits bei den vollständig-überwachten Experimenten eine lange Inferenzdauer bei dem Validierungsdatensatz zu erkennen ist (vgl. „Original" in Tabelle 5.3.2a), sollte die Inferenz verändert werden. In [ZG22a] wird die Beam-Search Implementierung der recht populären Huggingface Transformer Bibliothek [WDS+20] verwendet. Die Implementierung beinhaltet nur eine *Early-Stopping* Heuristik, welche die Beam-Search frühzeitig abbricht, falls es unwahrscheinlich ist, dass ein aktiver Beam der Suche besser werden kann als ein bereits gefundener Kandidat. Das führt jedoch dazu, dass die Beam-Breite in jedem Inferenzschritt voll verwendet wird. Dies geschieht auch dann, wenn beispielsweise nur eine Hypothese „wahrscheinlich" erscheint (bzgl. der Pseudowahrscheinlichkeit). Eine weitere Ineffizienz der Implementierung fällt bei der Verwendung von Batches auf: Wird die Inferenz mehrerer Hypothesen parallel durchgeführt, kann die Inferenz für einen einzelnen Datenpunkt vorzeitig beendet werden. Statt diesen Datenpunkt aus den weiteren Inferenzschritten zu entfernen, wird dieser in [ZG22a] mit Padding-Zeichen aufgefüllt und in der Beam-Search behalten. Dadurch belegt eine beendete Beam-Search weiterhin die volle Beam-Breite und verlängert aufgrund der vergrößerten Dimension des Eingabetensors die Zeit eines einzelnen Inferenzschrittes. Da die Inferenz bi-direktional durchgeführt wird, tritt dieser Nachteil auch bei einem einzelnen Datenpunkt auf.

Daher ist bei der Inferenz des gesamten nicht-annotierten Teils eine deutliche Verlängerung der Trainingsdauer zu erwarten. Wie in Abschnitt 2.5.1 beschrieben, ist die veränderte Beam-Search stark an [FAO17] angelehnt. Vor bzw. nach jedem Inferenzschritt werden die benötigten aktiven Beams bestimmt und inaktive Beams entfernt. Dadurch kann die Inferenz eines einzelnen Datenpunktes frühzeitig beendet werden, ohne dass dieser in der weiteren Inferenz in dem Eingabetensor enthalten ist. Weiterhin werden die beschriebenen Pruning-Methoden angewendet, um unwahrscheinliche Beams aus der Suche zu entfernen. Dadurch kann die aktive Menge an unvollständigen Hypothesen kleiner als die Beam-Breite sein, wodurch die Berechnungszeit verringert wird. Die Early-Stopping Heuristik aus [WDS+20] wird dabei weiterhin verwendet.

Metrik	Methode	CROHME			
		Train_{19}	14	16	19
Zeit	Original	37:40	3:20	4:15	4:07
	Neu	5:20	2:36	3:14	3:01
	$\text{Neu}_{\text{+Cons.}}$	5:14	2:38	3:15	3:04
ExpR_0	Original	87.5	52.5	53.4	59.2
	Neu	87.5	52.5	53.3	59.0
	$\text{Neu}_{\text{+Cons.}}$	87.5	52.3	53.1	58.8

(a)



(b)

Tabelle 5.3.2 & Abbildung 5.3.2: *Qualitative Analyse der Gesamtzeit der Inferenz auf verschiedenen Teilen des Datensatzes bei einem mit 50% der Annotationen vollständig-überwacht angelernten Netzwerk, wenn die verschiedenen Beam-Search Implementierungen verwendet werden. Dabei wird $\lambda_{const} = 0.15$ für das Constant Pruning genutzt.*

Für den Vergleich der beiden Implementierungen wird das CoMER Modell mit 50% der Trainingsdaten vollständig-überwacht angelernt. Dabei wird die Inferenzdauer der Validierung in jeder Epoche gemessen. Anschließend wird die Inferenzdauer auf den Trainings- sowie Testdaten aller CROHME Wettbewerbe gemessen. Zur Vergleichbarkeit wird das Training hintereinander auf demselben Computer und Grafikkarte durchgeführt. Die Ergebnisse davon sind Tabelle 5.3.2a dargestellt. Zusätzlich ist in Abbildung 5.3.2b die Zeit der Validierung für jede Epoche im Trainingsverlauf abgetragen. Dort ist die Verbesserung der Inferenzdauer auf dem Validierungsdatensatz zwischen den verschiedenen Beam-Search Implementierungen erkennbar. Die Implementierung von [WDS+20] ist durchwegs langsamer, wohingegen die Pruning-Methoden aus [FAO17] eine deutliche Verbesserung erzielen. Zusätzlich wird der Einfluss des Constant Prunings in den anfänglichen Epochen des Trainings sichtbar (vgl. Abschnitt 2.5.1). Ab Epoche 20 erzielt die Pruning Methode jedoch keine weitere Verbesserung. Der größte Unterschied tritt zu Beginn in der dritten Epoche auf: Die Implementierung aus [ZG22b] benötigt 60:20 Minuten (`MM:ss`), die veränderte Variante 55:35 bzw. 1:04 mit Constant Pruning. Hier wird der Vorteil des Constant Prunings deutlich, da die Beam-Search direkt abgebrochen wird, wenn die Konfidenz unterhalb des Mindestmaßes ist. Im Kontrast dazu wird deutlich, dass die Pruning Methoden aus [FAO17] erst bei einem gewissen Maß an Konfidenz eine deutliche Verbesserung der Inferenzdauer erzielen. Dies wird noch klarer beim Betrachten der vierten Epoche,

da dort die Inferenz insgesamt 57:04 (Original), 27:00 (Neu) bzw. 1:04 ($\text{Neu}_{+\text{Cons.}}$) andauert. Innerhalb der Epoche muss die Konfidenz so weit gestiegen sein, dass die Pruning Methoden aus [FAO17] stärker greifen. Dadurch ist die Inferenz deutlich schneller. In den anfänglichen Epochen kann dies jedoch die ExpR_0 auf dem Validierungsdatensatz verschlechtern. Deshalb wird das Constant Pruning nur am Anfang des Trainings verwendet und stufenweise reduziert. Für die weiteren Experimente der semi-überwachten Lernmethoden wird das Constant Pruning zu Beginn auf 0.15 gesetzt und stufenweise auf 0 reduziert. Bei Experimenten mit Schwellenwerten nahe oder unterhalb von 0.15 wird kein Constant Pruning verwendet, um auch für unsichere Vorhersagen eine vollständige Sequenz zu generieren. In Tabelle 5.3.2a ist zudem die Inferenzdauer auf den CROHME Trainingsdaten abgetragen, da diese für das Self-Training verwendet werden. Mithilfe der Pruning Methoden wird die Inferenzdauer auf rund 17% der originalen Implementierung reduziert. Da im Self-Training vom konvergierten Netzwerk weiter trainiert wird, ist dieser Unterschied auch für das gesamte Self-Training zu erwarten.
Jedoch darf die Erkennungsleistung bei der Anwendung der Pruning Methoden nicht deutlich sinken, um weiterhin bestmögliche Pseudolabel zu generieren. Wie Tabelle 5.3.2a zeigt, sinkt die Erkennungsleistung in manchen Fällen um bis zu 0.4 Prozentpunkte. Daher können die Pruning Methoden während der Generierung der Pseudolabel im Self-Training eingesetzt werden, ohne dass die ExpR_0 deutlich sinkt, wobei die Inferenzdauer deutlich verbessert wird.

5.4 SEMI-SUPERVISED LEARNING MIT SELF-TRAINING

Mit der veränderten Beam-Search Implementierung wird nun die in Abschnitt 4.3.1 beschriebene Self-Training Methode untersucht. Damit sollen die konvergierten Netzwerke des vorherigen Abschnitts 5.3 weiter trainiert werden, indem die nicht-annotierten Daten genutzt werden. Um das Potential der Methode einschätzen zu können, wird danach ein „perfektes“ Konfidenzmaß verwendet, welches die Konfidenz einer Hypothese abhängig von der Anzahl an Fehlern in der Vorhersage bestimmt.

5.4.1 *Self-Training mit CoMER Konfidenzmaß*

In den ersten Experimenten der semi-überwachten Lernmethode wird das Konfidenzmaß der CoMER Autoren aus [ZG22a] in Kombination mit verschiedenen Schwellenwerten verwendet (vgl. Abschnitt 4.3.1 und Formel 4.3.1). Dabei soll der Einfluss von fehlerbehafteten Pseudolabel und die Varianz der Schwellenwert-passierenden Datenpunkte untersucht werden.

Methode	50% CROHME			35% CROHME			15% CROHME		
	14	16	19	14	16	19	14	16	19
Baseline	52.5	53.4	**59.2**	48.3	51.0	54.5	37.1	39.1	40.5
$\tau = 0.985$	50.8	**54.4**	57.8	49.0	51.4	54.1	38.6	37.3	38.4
$\tau = 0.9$	52.9	53.6	58.1	49.6	**52.0**	55.6	41.0	38.5	41.6
$\tau = 0.85$	**53.0**	53.8	57.8	**50.0**	**52.0**	**55.9**	**41.1**	40.0	**42.3**
$\tau = 0.75$	50.9	52.9	59.0	48.3	50.2	54.6	38.8	40.0	42.0
$\tau = 0.5$	51.2	51.8	57.7	48.0	51.1	55.6	38.8	**40.3**	42.1
$\tau = 0.0$	51.8	53.3	58.5	48.1	49.9	53.0	39.1	39.2	41.9

Tabelle 5.4.1: *$ExpR_0$ (%) mehrerer mit der grundlegenden Self-Training Methode, unterschiedlichen Schwellenwerten und verschiedenen Teilen an Annotationen trainierte Netzwerke, verglichen mit den vollständig-überwacht angelernten Baselines.*

Die Ergebnisse sind in Tabelle 5.4.1 dargestellt und zeigen einen leichten Anstieg der Erkennungsleistung auf den verschiedenen Testdatensätzen der CROHME Wettbewerbe. Zu erkennen ist deutlich, dass ein niedriger Schwellenwert bessere Ergebnisse erzielt als ein hoher. Dies spricht dafür, dass das Netzwerk und die Lernmethodik eine gewisse Fehlertoleranz haben, sodass die höhere Anzahl an passierenden Pseudolabel zu einer Verbesserung der Erkennungsleistung beitragen. Das Aufnehmen aller nicht-annotierten Datenpunkte ($\tau = 0.0$) verschlechtert jedoch die Erkennungsleistung. Zudem erzielen unterschiedliche Schwellenwerte die besten Ergebnisse für verschiedene Teile des annotierten Datensatzes. Die Wahl des Schwellenwerts zum Erreichen beider Ziele ist daher scheinbar schwierig. Wünschenswert wäre hierbei jedoch ein einziger Schwellenwert, der für mehrere bzw. alle Experimente am Besten abschneidet. Wird $\tau = 0.985$ bzw. $\tau = 0.9$ mit den kleineren Schwellenwerten verglichen, fällt auf, dass die Erkennungsleistung tendenziell steigt. Mehr Fehler bzw. unsichere Hypothesen in das Training aufzunehmen scheint bis zu einem gewissen Punkt eine Verbesserung der $ExpR_0$ zu erreichen.
Im Allgemeinen sprechen die Ergebnisse dafür möglichst viele Datenpunkte mit möglichst wenig Fehlern in das Training mit aufzunehmen. Dies wird versucht mit der Kalibrierung in Abschnitt 5.5 zu vereinfachen und zu verbessern. Dadurch soll die Suche nach einem passenden Schwellenwert entsprechend des Datensatzes entfallen oder zumindest deutlich reduziert werden.

Abschließend lässt sich sagen, dass die grundlegende semi-überwachte Self-Training Methode in manchen Fällen eine kleine Verbesserung erzielt, diese jedoch unter den Erwartungen und Ergebnissen von z. B. FixMatch [SBL+20] bleibt. In [SBL+20] wird beispielsweise eine Verbesserung von rund 12 Prozentpunkten erreicht. Da die Ergebnisse jedoch in der Bildklassifikation statt in der Sequenzvorhersage erzielt werden, kann dies auch an den unterschiedlichen Problemdomänen liegen. Auf die beiden zentralen Forschungsfragen bezogen, kann die eingesetzte Self-Training Methode die ExpR_0 leicht verbessern, indem die nicht-annotierten Datenpunkte genutzt werden. Der Bedarf an annotierten Daten kann jedoch nur eingeschränkt reduziert werden. Bei unpassender Wahl des Schwellenwertes kann die ExpR_0 auch sinken.

5.4.2 *Orakel Konfidenzmaß*

Um eine Art „theoretisches Limit“ für die Methodik zu untersuchen, wird ein weiteres Konfidenzmaß mithilfe eines *Orakels* eingesetzt. Dabei werden die Annotationen des nicht-annotierten Teils im Orakel zwischengespeichert und im Training verwendet, um die tatsächlichen Fehler einer Vorhersage im Konfidenzmaß zu quantifizieren. Das Konfidenzmaß ergibt sich zu der Levenshtein Distanz [Lev66] und kann mithilfe von

$$\mathrm{ORACLE}(\mathbf{a}, \mathbf{p}) = \mathrm{lev}(\mathbf{a}, \mathbf{p}) \tag{5.4.1}$$

berechnet werden. Dabei bezeichnet $\mathbf{a}$ die Annotation des Datenpunktes und $\mathbf{p}$ die Vorhersage der Beam-Search. Die Levenshtein Distanz ist eine Editier-Distanz, welche die Summe an benötigten Modifikationen einer Zeichenkette zur Überführung in eine andere berechnet. Zwischen $(C, \_, 2)$ und $(c, \_, 2)$ beträgt diese zum Beispiel 1, da mit dem Ersetzen von C zu c die erste Zeichenkette in die zweite überführt werden kann. Neben dem Ersetzen wird auch das Hinzufügen und Entfernen von Zeichen beachtet. Durch diese Definition kann die erlaubte Fehleranzahl mit dem Schwellenwert begrenzt werden.

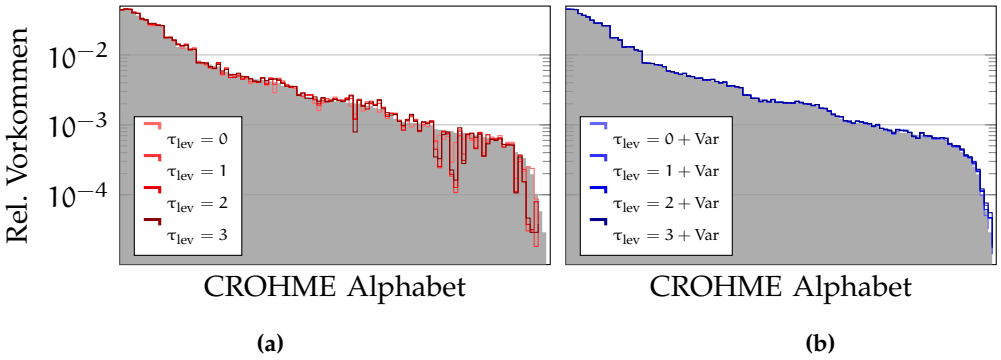
Neben der absoluten Fehleranzahl wird die Varianz der Schwellenwerte-passierenden Pseudolabel erhöht. Damit soll untersucht werden, ob durch eine künstliche Erhöhung der Varianz eine höhere Erkennungsrate erreicht werden kann, wenn die Fehleranzahl unverändert bleibt. Entsprechend des festgelegten Schwellenwerts wird für jede Schwellenwert-passierende Hypothese ein zufälliger nicht-annotierter Datenpunkt gezogen. In einer Epoche kann ein nicht-annotierter Datenpunkt nur einmal gezogen werden. Die Annotation des Orakels für den gezogenen Datenpunkt wird dann zufällig mit Fehlern versehen, damit die Fehleranzahl im resultierenden Pseudolabel

τ_{lev}	CROHME$_{19}$					
	50%		35%		15%	
		+Var		+Var		+Var
Baseline	59.2	59.2	54.5	54.5	40.5	40.5
0	**59.5**	61.1	**56.4**	**60.0**	**45.2**	53.8
1	58.3	**62.0**	55.8	**60.0**	44.3	54.7
2	**59.5**	61.2	55.9	59.5	44.5	54.9
3	58.1	61.4	55.0	59.5	43.6	**56.1**

Tabelle 5.4.2: *Vergleich der $ExpR_0$ (%) verschiedener Schwellenwerte im Self-Training mit Orakel Konfidenzmaß und verschiedenen Teilen an annotierten Datenpunkten. Gegenüberstellung mit einer anschließenden zufälligen Variation bei gleichbleibender absoluter Fehlerzahl und mit den ausschließlich vollständig-überwacht angelernten Netzwerken auf den CROHME$_{19}$ Testdaten.*

gleich bleibt. Die Fehleranzahl wird dabei mithilfe der Levenshtein Distanz bestimmt. Dadurch wird jede Schwellenwert-passierende Hypothese mit einem zufälligen nicht-annotierten Datenpunkt mit gleicher Fehleranzahl ersetzt. Für das Anwenden eines einzelnen Fehlers wird zufällig aus „Ersetzen“, „Entfernen“ und „Hinzufügen“ gezogen. Beim Ersetzen oder Hinzufügen wird das neue (fehlerhafte) Zeichen zufällig aus dem Alphabet des Datensatzes gezogen. Dadurch bleibt die absolute Fehlerzahl in den passierenden Pseudolabel gleich, wobei die Datenpunkte zufällig variieren. Die Variation der passierenden Pseudolabel wird dabei künstlich erhöht.

Die Ergebnisse des Trainings mehrerer Netzwerke mit unterschiedlichem Zugriff auf die Annotationen, welche mithilfe des Self-Trainings und beider Orakel-Methoden weiter trainiert werden, sind in Tabelle 5.4.2 dargestellt. Zunächst ist ersichtlich, dass die Erkennungsrate gegenüber der vollständig-überwacht angelernten Baseline in allen Fällen übertroffen wird. Zudem ist die relative Verbesserung umso größer, je weniger Trainingsdaten verfügbar sind. So erreicht „15% ohne Variation“ mit einem erlaubten Fehler eine Verbesserung von 4.7 Prozentpunkten, wohingegen bei „50% ohne Variation“ lediglich eine Verbesserung um 0.3 Prozentpunkte erzielt wird. Dies kann an der Nähe der Erkennungsrate zur 100% vollständig-überwacht angelernten Baseline liegen, da Tabelle 5.3.1a zeigt, dass der Unterschied in der Erkennungsrate zwischen 50% und 100% verwendeten Annotationen lediglich 3 Prozentpunkte beträgt. Weiterhin ist in der Tabelle 5.4.2 eine gewisse Fehlertoleranz gegenüber auftretenden



(a) **(b)**

Abbildung 5.4.1: *Verteilung über das CROHME Alphabet der Schwellenwert-passierenden Vorhersagen im Vergleich mit der grau hinterlegten Verteilung aller Annotationen der Trainingsdaten. Das* 15% *Netzwerk wird mit dem Orakel Konfidenzmaß ohne (a) bzw. mit (b) zufälliger Variation angelernt. Die Verteilung ist auf die Gesamtzahl an Zeichen der letzten* 5 *Epochen normalisiert.*

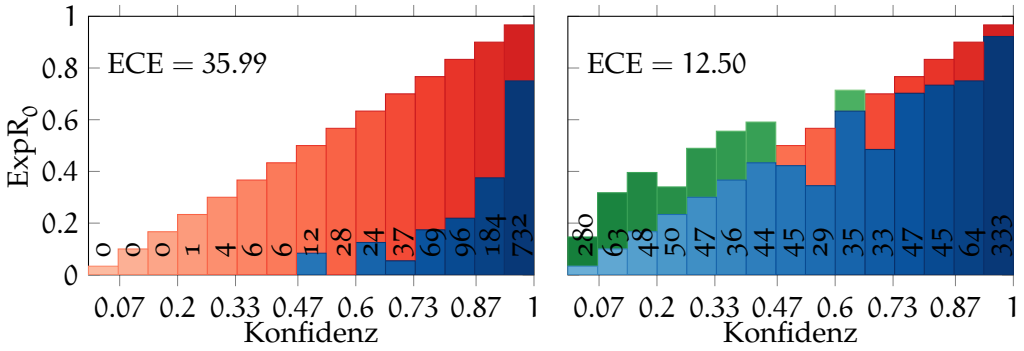
Fehlern in den Hypothesen zu erkennen. Ab drei erlaubten Fehlern ist die Erkennungsrate jedoch stets unterhalb von ein oder zwei erlaubten Fehlern (ohne zufällige Variation). Werden die weiteren Testdatensätze in Tabelle A.1.3 sowie A.1.4 mitbetrachtet, kann dort eine größere Fehlertoleranz erkannt werden. Im Allgemeinen sollte der Fehler aber möglichst klein gehalten werden. Zudem ist in der Spalte „50% ohne Variation“ zu erkennen, dass die Erkennungsrate bei zwei erlaubten Fehlern höher ist (59.5%) als bei einem (58.3%). Dies spricht dafür, eine möglichst große Anzahl an passierenden Pseudolabel zu erreichen. Selbiges kann in fast allen Fällen auch in Tabelle A.1.3 bzw. A.1.4 beobachtet werden. Da die Erkennungsrate bei drei erlaubten Fehlern jedoch meist leicht sinkt, sollten gleichzeitig möglichst viele Fehler vermieden werden. Die zufällige Variation zeigt, dass bei gleichbleibender Fehlerzahl eine möglichst diverse Menge an Datenpunkten die Erkennungsrate verbessert. In fast jedem direkten Vergleich erzielt die Anwendung von zufälliger Variation bessere Ergebnisse. Im Falle von 15% genutzten Annotationen kann die Erkennungsrate dadurch um 12.5 Prozentpunkte gesteigert werden. Wie „15% mit Variation“ zeigt, ist es dabei sogar besser mehr Fehler und damit mehr Datenpunkte zuzulassen, wenn dadurch die Variation verbessert wird.

Um dies noch genauer zu analysieren, wird in Abbildung 5.4.1 die Verteilung über dem CROHME Alphabet mehrerer Fehler-Schwellenwerte mit der Verteilung der gesamten Trainingsdaten verglichen. Zur Berechnung wird dabei die durchschnittliche Verteilung des Alphabets der letzten 5 Epochen gebildet. Dadurch soll nicht nur eine

Momentaufnahme der passierenden Pseudolabel einer einzelnen Epoche verglichen werden. Deutlich zu erkennen ist der Unterschied bei der Verwendung von zufälliger Variation (Abbildung 5.4.1b). Wird zufällige Variation verwendet, ist die durchschnittliche Alphabet-Verteilung deutlich näher an der des gesamten Datensatzes. Dies kann teilweise bei der Erhöhung der Fehlertoleranz auch ohne zufällige Variation beobachtet werden. Dadurch werden zwar mehr Fehler erlaubt, jedoch ist die Verteilung meist näher an der des vollständigen Datensatzes. Konkludent ist der Zusammenhang jedoch nicht, weswegen die Verbesserung der Erkennungsleistung wohl eher der Varianz an Eingabebildern zuzuschreiben ist. Zusätzlich sind vermutlich die absolute Anzahl an passierenden Datenpunkten, die darin auftretenden Schreibstile, Sequenzlängen und die Bildgrößen weitere Faktoren, welche die Erkennungsrate beeinflussen. Ähnliche Schlüsse werden dabei z. B. auch in [CTF+23] gezogen: Die Varianz in den Schwellenwert-passierenden Datenpunkte ist sehr wichtig, sodass in [CTF+23] alle Datenpunkte in das Training aufgenommen werden. Nur die Gewichtung bei der Optimierung des Netzwerks wird abhängig von der Konfidenz angepasst.

5.5 KALIBRIERUNG & OVERCONFIDENCE

Da die Ergebnisse der grundlegenden Self-Training Methode deutlich hinter dem Orakel-Experiment liegen, wird nun untersucht, ob eine Kalibrierung des Netzwerks eine Verbesserung erzielt. Zudem wird bei der Analyse der ECE Diagramme des vorherigen Experiments mit 35% genutzten Annotationen (vgl. Abschnitt 5.3) in Abbildung 5.5.1a deutlich, dass das Konfidenzmaß der CoMER Autoren keine gute Kalibrierung erzielt. Zu erkennen ist zum Einen eine starke Abweichung von der Diagonalen und zum Anderen eine deutliche Häufung fast aller Vorhersagen im obersten Intervall (732 von 1199). Anders ausgedrückt bedeutet dies, dass das Netzwerk für beinahe alle Hypothesen einen sehr hohen Konfidenzwert ausgibt, obwohl die ExpR_0 des Netzwerks auf den CROHME_{19} Testdaten bei rund 55% liegt. Durch die Kalibrierung soll der erwartete Kalibrierungsfehler minimiert werden, sodass keine bzw. eine möglichst kleine Abweichung von der Diagonalen in Abbildung 5.5.1 vorliegt. Zusätzlich sollen dadurch möglichst viele nicht-annotierte Datenpunkte in das Training zugelassen werden, während die gravierendsten Fehler vermieden werden. Darüber hinaus soll die Wahl des Schwellenwerts vereinfacht werden. Für die Verbesserung der Kalibrierung werden, wie in Abschnitt 4.3.2 beschrieben, verschiedene Konfidenzmaße, Temperature Scaling und LogitNorm untersucht. Die einzelnen Methoden können dabei alleine oder in Kombination betrachtet werden.



(a) **(b)**

Abbildung 5.5.1: *ECE-Diagramme eines mit 35% der CROHME$_{19}$ Trainingsdaten angelernten Netzwerks, welche das ORI (a) bzw. BIMIN (b) Konfidenzmaß nutzen und auf den CROHME$_{19}$ Testdaten evaluiert werden.*

5.5.1 *Konfidenzmaße*

Zunächst wird die Kalibrierung mithilfe verschiedener Konfidenzmaße untersucht. Formal sind die Konfidenzmaße in Abschnitt 4.3.2 definiert. Diese werden nun auf die Sequenzvorhersagen auf den CROHME$_{19}$ Testdaten von drei verschiedenen Netzwerken angewandt, welche mit unterschiedlichen Teilen der Trainingsdaten angelernt werden. In Tabelle 5.5.1 ist der erwartete Kalibrierungsfehler (ECE) sowie die Average Precision für jedes Konfidenzmaß abgetragen.

Zu erkennen ist deutlich, dass das BIMIN Konfidenzmaß die beste Kalibrierung bezüglich des erwarteten Kalibrierungsfehlers erzielt. Das zeigt auch das ECE-Diagramm in Abbildung 5.5.1b im Vergleich zur Abbildung 5.5.1a. Die Average Precision ist bei allen verwendeten Konfidenzmaßen vergleichbar. Das BIMULT Konfidenzmaß erreicht jedoch bei allen drei untersuchten Netzwerken die beste Average Precision. Die PR-Kurve des ORI, BIMIN und BIMULT Konfidenzmaßes in Abbildung A.1.1 bestätigen diese Ergebnisse: Die Kurven sind recht nah beieinander und unterscheiden sich nur minimal. Zudem ist in Tabelle 5.5.1 zu erkennen, dass der Kalibrierungsfehler bei weniger verwendeten Trainingsdaten deutlich größer wird, außer bei Verwendung des BIMIN oder BIMULT Konfidenzmaßes. Basierend auf diesen Ergebnissen scheint das BIMIN Konfidenzmaß eine bessere Wahl als das ORI Konfidenzmaß darzustellen. Es erzielt in den Experimenten den geringsten Kalibrierungsfehler und erreicht die zweithöchste Average Precision, wobei dies unabhängig von der Menge an Trai-

Konfidenzmaß	100% Baseline		35% Baseline		15% Baseline	
	ECE ↓	AP ↑	ECE ↓	AP ↑	ECE ↓	AP ↑
ORI	30.9	58.4	36.0	48.0	45.0	28.9
AVG	33.1	57.4	40.7	47.2	55.7	26.2
BIAVG	30.6	58.4	35.5	47.9	43.8	28.7
MIN	15.4	57.7	18.3	47.4	24.8	26.9
BIMIN	**8.8**	58.5	**12.5**	47.8	**10.6**	29.0
MULT	11.1	58.0	12.5	48.0	14.4	27.6
BIMULT	10.9	**58.9**	14.2	**48.3**	13.1	**29.7**

Tabelle 5.5.1: *Vergleich des ECE und der AP zwischen Konfidenzmaßen und mehreren trainierten Modellen auf den CROHME$_{19}$ Testdaten.*

ningsdaten zu beobachten ist. Die Kalibrierung des BIMIN Konfidenzmaßes ist also durchwegs besser verglichen mit dem ORI Konfidenzmaß und verbessert gleichzeitig die Sortierung der Vorhersageliste. Werden keine weiteren Kalibrierungsmethoden eingesetzt, sollte daher das BIMIN Konfidenzmaß in Self-Training Experimenten eingesetzt werden.

5.5.2 *Temperature Scaling*

Temperature Scaling kann nach [LNC+11] auch zur Verbesserung der Kalibrierung eingesetzt werden. Dafür wird der Parameter T mithilfe der Vorhersagen auf den CROHME$_{14}$ Testdaten und dem L-BFGS Algorithmus optimiert (vgl. Abschnitt 4.3.2). Für die Optimierung muss eine Loss-Funktion gewählt werden. Nach [LNC+11, Ple17] kann der CE-Loss auf dem Validierungsdatensatz zur Optimierung genutzt werden (vgl. Abschnitt 4.3.2). Andererseits kann jedoch auch der erwartete Kalibrierungsfehler (ECE) direkt optimiert werden. Eine weitere Möglichkeit ist die Summe beider Loss-Funktionen.
Die Optimierung mithilfe der CROHME$_{14}$ Testdaten wird auch in den weiteren Experimenten durchgeführt, sodass die ExpR$_0$ auf den CROHME$_{14}$ Testdaten nicht vollständig repräsentativ ist, weil die Testdaten indirekt während des Trainings verwendet werden. Da die Testdaten der anderen Jahre aber disjunkt sind, bleiben die Ergebnisse darauf repräsentativ.

Konfidenzmaß	T = 1		CE		ECE		CE + ECE	
	ECE ↓	AP ↑	ECE ↓	AP ↑	ECE ↓	AP ↑	ECE ↓	AP ↑
ORI	36.0	48.0	33.0	48.7	24.1	47.9	28.2	48.4
AVG	40.7	47.2	34.9	47.5	24.0	47.0	28.9	47.3
BIAVG	35.5	47.9	32.3	48.4	22.8	47.4	27.3	48.0
MIN	18.3	47.4	**6.0**	48.0	**12.5**	48.1	**5.9**	48.0
BIMIN	**12.5**	47.8	9.8	48.4	20.5	**48.2**	16.1	48.4
MULT	12.5	48.0	9.8	48.8	35.3	48.1	26.0	48.6
BIMULT	14.2	**48.3**	25.7	**49.2**	46.2	**48.2**	39.3	**48.9**

Tabelle 5.5.2: *Vergleich des ECE und der AP zwischen Konfidenzmaßen und verschiedenen Optimierungsvarianten des Temperature Scaling Parameters* T *(vgl. Abschnitt 4.3.2) auf den CROHME*$_{19}$ *Testdaten eines mit* 35% *der Annotationen angelernten Netzwerks.*

Die Ergebnisse sind in Tabelle 5.5.2 dargestellt und beinhalten die verschiedenen Optimierungsvarianten des Parameters T für ein mit 35% Trainingsdaten angelerntes Netzwerk. Zum Vergleich sind die Ergebnisse ohne Temperate Scaling (T = 1) in der linken Spalte abgetragen (vgl. Tabelle 5.5.1). Auch hier erreicht BIMIN bzw. MIN die beste Kalibrierung bezüglich des erwarteten Kalibrierungsfehlers und BIMULT hat erneut die höchste Average Precision. Durch Temperature Scaling wird die Kalibrierung bei fast allen untersuchten Konfidenzmaßen verbessert, lediglich die Kalibrierung des BIMULT Konfidenzmaßes wird durch Temperature Scaling verschlechtert. Beim Betrachten der zugehörigen ECE-Diagramme wird der Grund deutlich (vgl. Abbildung A.1.2): Das Netzwerk neigt mit Temperature Scaling zur Underconfidence. Da das Temperature Scaling den Ausgabevektor des Netzwerks skaliert und der Konfidenzwert dadurch meist sinkt, wird der aggregierte Konfidenzwert durch die Multiplikation aller einzelnen Konfidenzwerte der Zeichen deutlich kleiner.
Die Art der Optimierung hat auch einen großen Effekt auf die Kalibrierung des Netzwerks. Durch die Kombinationen beider Loss-Funktionen kann insgesamt zwar der geringste Kalibrierungsfehler mit dem MIN Konfidenzmaß erreicht werden, jedoch schneiden alle anderen Konfidenzmaße deutlich schlechter ab. Die Verwendung des ECE in der Loss Funktion hat sich jedoch während der Optimierung des Parameters T als instabil herausgestellt, sodass T stellenweise ins Negative divergiert. Daher sollte der Cross-Entropy Loss zur Optimierung verwendet werden. Dann lässt sich festhalten, dass das MIN, BIMIN bzw. MULT Konfidenzmaß die besten Kalibrierungsergebnisse erzielt.

5.5.3 *LogitNorm*

Als zweite Methode zur Kalibrierung wird LogitNorm untersucht. Dabei kann und soll die Methode mit Temperature Scaling kombiniert werden [WXC+22]. Einerseits wird erneut die Kombination mit einem Konfidenzmaß evaluiert und andererseits wird die Wahl des Hyperparameters τ_{LN} untersucht, welcher in LogitNorm für die Normierung des Ausgabevektors verwendet wird (vgl. Formel 3.3.2).

Zur Bestimmung von τ_{LN} werden, analog zu [WXC+22], mehrere Werte im Bereich $[0.02, 0.5]$ ausgetestet und die $ExpR_0$ sowie die Kalibrierungsleistung miteinander verglichen. Dies ist in Abbildung 5.5.3b dargestellt, wobei sowohl die Kalibrierung als auch die Erkennungsleistung ab $\tau_{LN} \geqslant 0.1$ stark sinkt. Die beste $ExpR_0$ wird mit $\tau_{LN} = 0.075$ erreicht (55.2%) und der niedrigste erwartete Kalibrierungsfehler (4.5) kann in dem Experiment mit $\tau_{LN} = 0.0625$ erzielt werden. Diese Ergebnisse decken sich annähernd mit denen aus [WXC+22], welche für die beste Kalibrierung $\tau_{LN} = 0.04$ nutzen und auch eine Verschlechterung für höhere Werte von τ_{LN} beobachten.
Tabelle 5.5.3a verdeutlicht, dass die Kombination mit Temperature Scaling eine wesentliche Verbesserung der Kalibrierung erzielt. Dadurch wird der Kalibrierungsfehler fast aller Konfidenzmaße stark reduziert, wobei der kleinste erwartete Kalibrierungsfehler (4.5) niedriger ist als der des Temperature Scaling Experiments (5.9, vgl. Tabelle 5.5.2). Dies deckt sich mit [WXC+22], welche bei einer geeigneten Wahl von τ_{LN} eine kleine Verbesserung der Kalibrierung gegenüber Temperature Scaling erreichen.

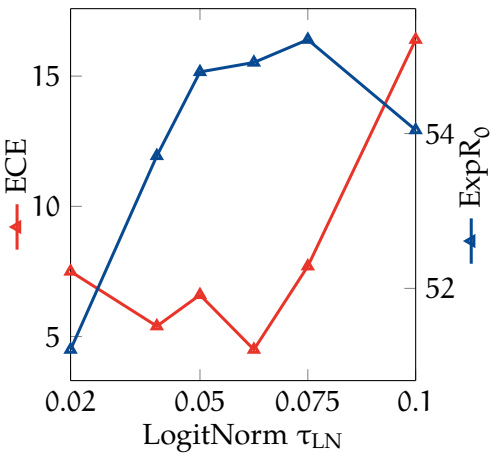
Zusammenfassend ist LogitNorm also eine gute Alternative, um die Kalibrierung des Netzwerks zu verbessern, wobei eine vorherige Suche nach einem passenden Hyperparameter sowie Konfidenzmaß notwendig ist. Bei einer passenden Wahl kann die Kalibrierung jedoch besser sein als die exklusive Verwendung von Temperature Scaling. Wird LogitNorm verwendet, sollte dies aber immer mit Temperature Scaling kombiniert werden, da die Kalibrierung sich sonst meist drastisch verschlechtert.

5.5.4 *Self-Training mit Kalibrierung*

Im Folgenden werden die Kalibrierungsmethoden in das Self-Training integriert und die vorherigen Experimente aus Abschnitt 5.4 wiederholt. Erneut werden dafür drei verschiedene Netzwerke mithilfe des nun kalibrierten Self-Trainings weiter angelernt. Da Temperature Scaling und ein passendes Konfidenzmaß bereits gute Kalibrierungsergebnisse erzielen, wird das BIMIN Konfidenzmaß mit Temperature Scaling und

Maß	LN $\tau_{LN} = 0.0625$		LN + TS $\tau_{LN} = 0.0625$	
	ECE ↓	AP ↑	ECE ↓	AP ↑
ORI	33.8	41.7	31.2	46.8
AVG	**33.3**	40.8	33.1	44.8
BIAVG	33.8	41.2	30.4	46.5
MIN	45.8	44.8	**4.5**	45.0
BIMIN	46.8	**45.0**	14.3	46.6
MULT	52.4	38.6	16.1	45.7
BIMULT	52.5	38.8	30.1	**47.0**

(a)



(b)

Tabelle 5.5.3 & Abbildung 5.5.2: *Vergleich der Kalibrierung zwischen verschiedenen Konfidenzmaßen (a) bzw. LogitNorm Normierungsfaktoren (b). In (b) ist zusätzlich die $ExpR_0$ (%) auf den $CROHME_{19}$ Testdaten abgetragen. Dafür wird ein Netzwerk mit 35% der Trainingsdaten vollständig-überwacht angelernt.*

Cross-Entropy Loss Optimierung verwendet. Zum Vergleich werden drei weitere Netzwerke mit LogitNorm sowie $\tau_{LN} = 0.0625$ vollständig-überwacht angelernt und dann mit dem Self-Training, dem MIN Konfidenzmaß und den verschiedenen Schwellenwerten weiter trainiert. Die Experimente sollen dabei drei Fragen beantworten: Kann mithilfe der Kalibrierung eine bessere $ExpR_0$ durch die semi-überwachte Lernmethode erreicht werden? Kann ein einzelner Schwellenwert die besten Ergebnisse erzielen? Sollte LogitNorm oder Temperature Scaling für die weiteren Experimente verwendet werden?

Die Ergebnisse der Experimente mit verschiedenen Schwellenwerten sind in Tabelle 5.5.4 dargestellt und zeigen meist eine größere Verbesserung der $ExpR_0$ gegenüber den Experimenten ohne Kalibrierung (vgl. Tabelle 5.4.1). Im Falle von 35% genutzten Annotationen wird eine Erkennungsrate von 56.1% (TS) bzw. 55.7% (LN+TS) erreicht. Ohne Kalibrierung wird dahingegen 55.9% erzielt. Bei 15% verwendeten Annotationen wird 43.3% bzw. 46.0% statt 42.1% erzielt. Hier ist also eine deutliche Verbesserung gegenüber der nicht kalibrierten Methode zu erkennen. Bei 50% wird in den vorherigen Experimenten ohne Kalibrierung keine Verbesserung erzielt, wohingegen hier

Methode	CROHME$_{19}$					
	50%		35%		15%	
	TS	LN + TS	TS	LN + TS	TS	LN + TS
Baseline	58.9	**60.2**	55.5	54.0	40.5	37.8
$\tau = 0.95$	58.8	59.0	54.0	54.3	36.3	38.5
$\tau = 0.5$	57.0	59.6	54.1	54.7	39.3	44.1
$\tau = 0.125$	57.8	59.5	54.5	**55.7**	42.4	45.0
$\tau = 0.05$	**59.3**	59.4	**56.1**	55.4	**43.3**	**46.0**

Tabelle 5.5.4: *ExpR$_0$ (%) auf den CROHME$_{19}$ Testdaten mehrerer, mit kalibriertem Self-Training, unterschiedlichen Schwellenwerten und verschiedenen Teilen an Annotationen trainierte Netzwerke, verglichen mit den vollständig-überwacht angelernten Baselines. Für die Kalibrierung wird Temperature Scaling sowie LogitNorm mit Temperature Scaling verwendet.*

eine kleine Verbesserung von 0.1 Prozentpunkten erreicht wird. Die Ergebnisse auf dem CROHME$_{14}$ bzw. CROHME$_{16}$ Testdatensatz sind in Tabelle A.1.5 und A.1.6 dargestellt und zeigen ein ähnliches Bild. Mithilfe der Kalibrierung kann also bei den hier untersuchten Hyperparametern eine Erhöhung der Erkennungsrate erreicht werden. Hinsichtlich der ersten Frage kann festgehalten werden, dass mithilfe der Kalibrierung eine weitere Verbesserung der Methodik sowie Erkennungsrate erreicht werden kann.

Zudem können deutlich kleinere Schwellenwerte verwendet werden, da die Hypothesen durch die Kalibrierung nun breiter im Konfidenzraum verteilt sind statt sich im letzten Konfidenzintervall zu häufen (vgl. Abbildung A.1.2 bzw. A.1.3). Bei der Verwendung von Temperature Scaling wird die größte Verbesserung beinahe immer bei einem Schwellenwert von $\tau = 0.05$ erzielt. Dies deutet darauf hin, dass die kalibrierte Methodik numerisch stabiler ist und fehlerhafte Hypothesen eher vermieden werden. Wie in der Tabelle zu erkennen, ist eine größere Variation durch einen kleineren Schwellenwert hilfreich. Je kleiner der Schwellenwert ist, desto besser werden die Erkennungsraten in den meisten Fällen. Die Erkennungsrate könnte ggf. noch weiter verbessert werden, wenn Schwellenwerte $\tau < 0.05$ untersucht werden. Da die vorherigen Experimente jedoch eine Verschlechterung der Erkennungsrate bei $\tau = 0.0$ gezeigt haben, fällt die Erkennungsrate vermutlich ab einem Schwellenwert $0.0 < \tau < 0.05$.

Ähnliche Ergebnisse sollten aber auch ohne die Anwendung der Kalibrierungsmethoden reproduzierbar sein, weil die Vorhersagen nicht stark von der Kalibrierung beeinflusst werden. Jedoch müsste dafür der Schwellenwert geschickt im Häufungsgebiet der unkalibrierten Konfidenzen gewählt werden. Zudem wäre die Menge an Schwellenwert-passierenden Hypothesen ohne Kalibrierung vermutlich deutlich schwankender, da sich der Konfidenzwert für Hypothesen nahe dem Schwellenwert nur minimal ändern muss, um ober- bzw. unterhalb zu liegen. Zudem kann die Menge in einem festen Bereich um den Schwellenwert deutlich größer sein als bei der Verwendung von Kalibrierung, da die Hypothesen durch die Kalibrierung besser im Konfidenzraum verteilt sind (vgl. Abbildung A.1.2 bzw. A.1.3). Mithilfe der Kalibrierungsmethoden kann die Menge an Hypothesen nahe dem Schwellenwert reduziert werden, sodass die Methode unempfindlicher für numerische Schwankungen von Epoche zu Epoche sein sollte. Daher kann die Menge an verworfenen Pseudolabel besser gesteuert werden, wenn das Netzwerk kalibriert ist. Insgesamt wird dadurch die Wahl des Hyperparameters deutlich vereinfacht, da ein einzelner Schwellenwert die besten Ergebnisse erzielt oder nahe des besten Ergebnisses ist. Die Ergebnisse variieren deutlich weniger von Experiment zu Experiment. Die zweite Frage kann daher mit der vorherigen Analyse des Schwellenwertes beantwortet werden: Ein einzelner Schwellenwert erreicht für Temperature Scaling gute Ergebnisse über alle drei Netzwerke. Zudem spricht die Verteilung der Vorhersagen im Konfidenzraum für ein stabileres semi-überwachtes Anlernen. Mithilfe der Kalibrierung kann daher ein Experiment-unabhängiger Schwellenwert verwendet werden, welcher vermutlich nur von der allgemeinen Problemdomäne abhängt.

Schließlich kann zur Beantwortung der letzten Frage ein leichter Unterschied zwischen LogitNorm und Temperature Scaling in Tabelle 5.5.4 erkannt werden. Auf der einen Seite schneidet die LogitNorm Baseline leicht schlechter ab, wenn das Netzwerk nur Zugriff auf 15% bzw. 35% der Annotationen hat. Werden auf der anderen Seite 50% der Annotationen benutzt, übertrifft LogitNorm die bisherigen vollständig-überwacht angelernten Netzwerke. LogitNorm scheint daher ab einem gewissen Punkt eine bessere ExpR_0 zu erreichen als die mit Cross-Entropy Loss trainierten Netzwerke. Der relative Zugewinn an Erkennungsleistung ist aber vergleichbar zu den mit Temperature Scaling kalibrierten Netzwerken. Eine Ausnahme davon stellt das Experiment mit 15% der annotierten Trainingsdaten dar, da dort eine Verbesserung von rund 8 statt 3 Prozentpunkten erreicht wird. Im Allgemeinen kann aber weder LogitNorm noch Temperature Scaling als bessere Kalibrierungsmethode für das Self-Training angesehen werden. Beide Kalibrierungsmethoden erzielen jedoch bessere und stabilere Ergebnisse als die Experimente ohne Kalibrierung. Da LogitNorm jedoch bei

wenig annotierten Daten nicht die erwartete $ExpR_0$ erreicht, sollte abhängig von der Problemdomäne und dem Datensatz der passendere Ansatz gewählt werden. Hinzu kommt eine vorherige Suche und Analyse des Normierungsfaktors für LogitNorm, falls diese Methode eingesetzt wird.

Zusammenfassend können die Ergebnisse der Experimente Teile der beiden zentralen Forschungsfragen beantworten: Mithilfe des kalibrierten Self-Trainings können die nicht-annotierten Daten zur Verbesserung der Erkennungsleistung für handschriftliche mathematische Formeln genutzt werden. Je weniger annotierte Datenpunkte im Training verfügbar sind, desto größer ist der Zugewinn an $ExpR_0$ durch die semi-überwachte Lernmethode. Des Weiteren kann der Datenaufwand reduziert werden, wobei die Erkennungsleistung eines mit allen Annotationen vollständig-überwacht angelernten Netzwerks nicht erreicht wird.

5.6 VERÄNDERUNG UND ERWEITERUNG DER METHODIK

Da die grundlegende Methodik mithilfe der Kalibrierung und des passenden Schwellenwertes konsistent bessere Ergebnisse erzielt, werden im Folgenden mögliche Erweiterungen der semi-überwachten Lernmethode untersucht. Dafür wird einerseits ein Vortraining mit synthetischen Daten und andererseits die Partial Labeling Heuristik verwendet. Die restliche Methodik bleibt dabei unverändert. Um die Stabilität des Schwellenwertes der vorherigen Experimente weiter zu untersuchen, werden erneut unterschiedliche Schwellenwerte getestet und miteinander verglichen. Zur weiteren Untersuchung der zweiten Forschungsfrage — inwieweit der annotierte Datenaufwand reduziert werden kann — wird die Teilmenge an verfügbaren annotierten Trainingsdaten noch weiter verkleinert.

5.6.1 *Vortrainieren mit synthetischen Daten*

In den vorherigen Experimenten kann keine Konvergenz unterhalb von 15% verwendeten Annotationen erreicht werden. Diese Menge kann jedoch mithilfe des synthetischen Vortrainings auf 2.5% (220 annotierte Datenpunkte statt 8 836) reduziert werden.
Ein Netzwerk wird dabei in drei Schritten trainiert: Zunächst werden die synthetisch generierten Daten des NTCIR-12 MathIR Datensatzes genutzt und in einem vollständig-überwachten Training verwendet (vgl. Abschnitt 5.1.2). Das auf den synthetischen Daten konvergierte Netzwerk wird dann als Zwischenpunkt verwendet. Im zweiten Schritt wird ein Teil der CROHME Trainingsdaten genutzt, um ausgehend vom Zwi-

Methode	15% CROHME			2.5% CROHME		
	14	16	19	14	16	19
Ohne Vortraining	37.1	39.1	40.5	—	—	—
Syn. Baseline	43.8	46.6	49.6	19.3	22.0	21.9
$\tau = 0.25$	46.3	47.2	**52.5**	25.8	26.1	27.3
$\tau = 0.125$	**46.8**	46.5	50.7	27.6	28.2	30.4
$\tau = 0.05$	46.1	**48.3**	51.7	**28.7**	**30.6**	**32.2**

Tabelle 5.6.1: *$ExpR_0$ (%) von vollständig- bzw. semi-überwacht angelernten Netzwerken ggf. mit einem vorherigen synthetischen Vortraining, welche auf den verschiedenen CROHME Testdatensätzen evaluiert werden.*

schenpunkt eine weitere Konvergenz zu erreichen. Schließlich wird das kalibrierte Self-Training aus dem vorherigen Abschnitt 5.5 wiederholt. Da Temperature Scaling und LogitNorm ähnliche Kalibrierungsergebnisse erzielt haben, wird im Folgenden nur Temperature Scaling untersucht.

Wie in Tabelle 5.6.1 zu sehen, ist durch das synthetische Vortraining eine deutliche Verbesserung der $ExpR_0$ zu erkennen. Der Unterschied zwischen der Baseline mit bzw. ohne synthetisches Vortraining beträgt bis zu rund 9 Prozentpunkte. Die Experimente mit 2.5% der annotierten Trainingsdaten zeigen zudem erneut die Wichtigkeit der Varianz. Bei 2.5% genutzten Annotationen kann in allen Fällen die beste Erkennungsrate durch die kalibrierte Self-Training Methode und einen Schwellenwert von $\tau = 0.05$ erzielt werden. Auch bei 15% verwendeten Annotationen kann mithilfe des gleichen Schwellenwertes eine Verbesserung erzielt werden, jedoch erreichen die weiteren untersuchten Schwellenwerte stellenweise bessere Ergebnisse. Dennoch ist der Abstand zur höchsten Erkennungsleistung gering. Dies spricht erneut für die Stabilität des Schwellenwertes, auch wenn dieser nicht immer die optimale Wahl ist. Werden 15% der Annotationen verwendet, ist eine Fehlervermeidung anscheinend meist wichtiger als die größere Varianz. Dies verstärkt nochmals die Notwendigkeit beide Ziele gleichzeitig zu erreichen.
Vermutlich kann durch das Vortraining ein implizites Sprachmodell von validen LaTeX Formeln angelernt werden, sodass das Netzwerk tendenziell wohlgeformte Sequenzen generiert. Diese Vermutung wird durch Abbildung A.1.5 unterstützt: Durch das synthetische Vortraining werden deutlich längere Sequenzen korrekt vorhergesagt. Zudem ist die StructRate mit 75.9% gegenüber 61.5% deutlich erhöht (15% genutzte

Annotationen, evaluiert auf den CROHME_{19} Testdaten), was auch für ein generell verbessertes implizites Sprachmodell spricht. Die StructRate gibt dabei den Anteil an Sequenzen an, welche eine korrekte Struktur im MathML Baum haben. In der Berechnung fließt aber im Gegensatz zur ExpR_0 die korrekte Vorhersage eines Blattknotens nicht mit ein. Dadurch kann z. B. ein Zeichen falsch erkannt werden, wenn die sonstige Struktur der Formel korrekt ist. In der StructRate wird daher der Aufbau und die Verschachtelung der Formel betrachtet.

Bezogen auf die Forschungsfragen kann mithilfe des synthetischen Vortrainings der Bedarf an annotierten Daten weiter reduziert werden oder die Erkennungsleistung verbessert werden. Bei 100% genutzten CROHME Trainingsdaten kann die ExpR_0 durch das synthetische Vortraining von 63.4% auf 64.1% verbessert werden. Falls weniger der annotierten Daten verwendet werden, vergrößert sich die relative Verbesserung durch das synthetische Vortraining. Je weniger annotierte Daten vorliegen, desto wichtiger ist also das synthetische Vortraining.

5.6.2 *Fehlervermeidung durch Partial Labeling*

Als abschließende Erweiterung der semi-überwachten Lernmethode wird die Partial Labeling Heuristik statt des harten Schwellenwertes eingesetzt. Dadurch sollen Sequenzen unterhalb des Schwellenwertes nicht vollständig verworfen werden, sondern in Teilen im Training genutzt werden. Die Heuristik ist dabei in Abschnitt 4.3.3 formal beschrieben und die vorgestellten Profile werden in den folgenden Experimenten evaluiert. Dafür werden die Baseline Netzwerke der Experimente zum synthetischen Vortraining verwendet. Die Netzwerke werden also synthetisch vortrainiert und dann mithilfe von 15% bzw. 2.5% der CROHME Trainingsdaten vollständig-überwacht weiter angelernt. Ausgehend von diesem Zwischenpunkt wird die Self-Training Methode mit Partial Labeling innerhalb eines dritten Trainingsschrittes angewendet. Dabei werden erneut verschiedene Schwellenwerte untersucht, die den Start der Partial Labeling Heuristik steuern.

Wie Tabelle 5.6.2 zeigt, kann mithilfe der Partial Labeling Heuristik eine weitere deutliche Verbesserung der ExpR_0 erzielt werden. Im Vergleich zu den vorherigen Experimenten in Tabelle 5.6.1, erzielen alle getesteten Hyperparameter der Partial Labeling Heuristik bessere Ergebnisse, wenn 15% der annotierten Trainingsdaten genutzt werden. Verglichen mit dem vollständig-überwachten Anlernen des ersten Experiments kann mit dem erweiterten Self-Training daher eine Verbesserung von

Methode	CROHME$_{19}$					
	15%			2.5%		
	Low	*Med*	*High*	*Low*	*Med*	*High*
Ohne Vortraining	40.5	40.5	40.5	—	—	—
Syn. Baseline	49.6	49.6	49.6	21.9	21.9	21.9
$\tau = 0.5$	54.0	54.3	53.8	26.2	24.7	25.8
$\tau = 0.25$	53.5	54.1	53.2	27.0	25.4	28.5
$\tau = 0.125$	54.0	**56.0**	**55.0**	**30.6**	**30.7**	29.6
$\tau = 0.075$	**54.2**	53.5	55.0	27.9	29.2	**30.9**

Tabelle 5.6.2: *$ExpR_0$ (%) von vollständig- bzw. semi-überwacht angelernten Netzwerken ggf. mit einem vorherigen synthetischen Vortraining, welche die Partial Labeling Heuristik mit den Profilen aus Abschnitt 4.3.3 nutzen und auf dem CROHME$_{19}$ Testdatensatz evaluiert werden.*

insgesamt 15.5 Prozentpunkten erzielt werden, welches einer relativen Verbesserung von 38% entspricht. Die Fehlervermeidung kann mithilfe der PR-Kurve in Abbildung A.1.4 veranschaulicht werden: Unterhalb des Schwellenwertes werden Hypothesen beim unsichersten Zeichen aufgeteilt, um einen möglichen Fehler zu vermeiden. Dadurch steigt der Anteil an korrekten Teilsequenzen und die Average Precision über die getesteten Hypothesen. Im Training können jedoch vollständige und fehlerbehaftete Sequenz wichtiger für das Training sein, da bei 2.5% verfügbaren Annotationen die besten Ergebnisse nicht übertroffen werden können. In diesem Falle scheint es also besser zu sein, fehlerbehaftete aber dafür vollständige Sequenzen im Training zu nutzen. Eventuell werden dadurch unsichere Zeichen angelernt, die ansonsten meist durch die Partial Labeling Heuristik beim Aufteilen verworfen werden.

Hinsichtlich der ersten Forschungsfrage kann die mögliche Verbesserung durch die Partial Labeling Heuristik vergrößert werden. Die besten Ergebnisse werden dabei mit $\tau = 0.125$ und dem „Med“ Profil für 15% der annotierten Trainingsdaten erzielt. Werden nur 2.5% der annotierten Trainingsdaten verwendet, erreicht das „High“ Profil in Kombination mit $\tau = 0.075$ die besten Ergebnisse. Diese Ergebnisse stützen nochmals die vorherigen Vermutungen, dass eine möglichst hohe Varianz bei möglichst guter Fehlervermeidung die besten Ergebnisse erreicht (vgl. Abschnitt 5.4.2). Durch das „High“ Profil wird keine Vorhersage pauschal ausgelassen, sondern mithilfe der Heuristik aufgeteilt. Bei Betrachtung der passierenden Pseudolabel für jede Epoche ist auch

genau das der Fall: In beinahe jeder Epoche werden 8 612 von 8 616 in das Training aufgenommen („High“ Profil). Bei 15%, $\tau = 0.125$ und Verwendung des „Med“ Profils werden zwischen 7 300 und 7 400 von insgesamt 7 506 nicht-annotierten Datenpunkte in das Training aufgenommen. Beim Vergleich mit den Experimenten ohne Partial Labeling Heuristik wird deutlich, dass nicht nur mehr Vorhersagen den Schwellenwert bzw. die Heuristik passieren, sondern die absolute Fehleranzahl vermutlich geringer ist: Während des Self-Trainings des synthetisch vortrainierten und mit 15% vollständig-überwacht angelernten Netzwerks, werden zwischen 6 400 und 7 400 Vorhersagen in das Training aufgenommen, wobei ab Epoche 120 durchgehend über 7 300 Vorhersagen den Schwellenwert passieren. Da die Erkennungsleistung durch die Partial Labeling Heuristik jedoch deutlich übertroffen wird, bleibt die Fehleranzahl als Unterschied zwischen den beiden Experimenten übrig.

Mithilfe der Partial Labeling Heuristik erreicht das synthetisch vortrainierte, mit 15% vollständig-überwacht angelernte und mit der erweiterten Self-Training Methode trainierte Netzwerk insgesamt 56.0% (statt 40.5%, vgl. Tabelle 5.3.1a) und erreicht damit ungefähr die Erkennungsleistung eines mit 50% vollständig-überwacht angelernten Netzwerks (59.2%, vgl. Tabelle 5.3.1a). Hinsichtlich der ersten Forschungsfrage können die nicht-annotierten Daten für eine Verbesserung der Erkennungsleistung verwendet werden. Das erweiterte Self-Training erzielt dabei eine Verbesserung von bis zu 15.5 Prozentpunkten. Die zweite Forschungsfrage bezüglich der Reduktion des Datenaufwands kann damit auch besser beantwortet werden: Mithilfe des synthetischen Vortrainings und der Partial Labeling Heuristik ist es möglich den Datenaufwand in einem gewissen Maße zu reduzieren. Besonders wenn wenig annotierte Daten vorliegen, erreichen die vorgestellten Methoden gute Ergebnisse. Jedoch muss ein Mindestmaß an Annotationen vorliegen, damit die Partial Labeling Heuristik bessere Ergebnisse erzielt als ein harter Schwellenwert.

5.6.3 *Testdaten als zusätzliche nicht-annotierte Trainingsdaten*

Nach der Evaluation der inkrementell aufgebauten Self-Training Methode auf den CROHME Trainingsdaten wird im Folgenden untersucht, ob sich die Erkennungsrate im „Produktionsbetrieb“ noch weiter verbessern lässt. Die Testdaten eines Netzwerks sollten idealerweise die Daten im Produktionsbetrieb widerspiegeln und es ist wünschenswert, dass das Netzwerk besonders auf diesen eine möglichst hohe ExpR_0 erreicht. Die Datenpunkte aus dem Produktionsbetrieb sollen dafür als nicht-annotierte Daten in das Training aufgenommen werden. Mithilfe des Self-Trainings kann ein

Methode	100% CROHME			50% CROHME			15% CROHME		
	14	16	19	14	16	19	14	16	19
Baseline	57.1	60.0	63.4	52.5	53.4	59.2	37.1	39.1	40.5
Syn. Baseline	56.3	59.5	64.1	54.9	57.0	61.3	43.8	46.6	49.6
$\tau = 0.05$	61.3	65.4	**71.2**	59.3	**64.1**	**67.6**	48.1	53.0	55.2
$\tau_H = 0.125$	**62.6**	**66.6**	70.4	59.3	62.8	66.7	49.6	54.3	56.2
$\tau_M = 0.125$	61.7	66.2	70.7	**61.0**	63.0	67.4	**51.3**	**54.8**	**56.4**

Tabelle 5.6.3: *$ExpR_0$ (%) mehrerer mit der erweiterten Self-Training Methode trainierte Netzwerke, welche unterschiedliche Teile an annotierten Daten und Hyperparameter für die Partial Labeling Heuristik nutzen (vgl. Abschnitt 4.3.3). Die Testdaten der CROHME Datensätze werden ohne die Annotationen als zusätzliche Trainingsdaten verwendet. τ_H bzw. τ_M bezeichnen den Schwellenwert des Partial Labeling Profils „High“ bzw. „Mid“.*

Netzwerk mit den annotierten Trainingsdaten vortrainiert werden und anschließend in einem gewissen Rahmen auf das gewünschte Einsatzgebiet angepasst werden. Die CROHME Testdaten werden dabei als Daten des Produktionsbetriebs angesehen und ohne die Annotationen im Training verwendet.
Der Grundgedanke stammt dabei aus der Transduktion [VC74]. Nach Vapnik [Vap06, 465] soll kein generalisierendes Modell erstellt werden, welches mithilfe der Testdaten evaluiert wird. Stattdessen soll ein auf die Testdaten zugeschnittenes Modell erstellt werden, welches auf diesen besonders gute Ergebnisse erzielt, aber eventuell schlechter oder gar nicht generalisiert. Dies ist jedoch nicht vollständig auf Neuronale Netze übertragbar, da das erstellte Modell ein induktives Modell bleibt. Der Grundgedanke der Transduktion kann jedoch übertragen werden.

Ausgehend von den synthetisch vortrainierten und mit Teilmengen der Annotationen vollständig-überwacht angelernten Netzwerke, wird die Self-Training Methode angewendet. Dabei bestehen die nicht-annotierten Datenpunkte aus den restlichen Trainingsdaten sowie den Testdaten der CROHME Datensätze, wobei die Annotationen nur während der Validierung verfügbar sind. Methodisch wird das kalibrierte Self-Training aus Abschnitt 5.5 mit $\tau = 0.05$ sowie die Partial Labeling Heuristik aus dem vorherigen Abschnitt 5.6.2 verwendet und miteinander verglichen. Dabei wird das „Med“ und „High“ Profil mit einem Schwellenwert von $\tau = 0.125$ untersucht.

Die Ergebnisse in Tabelle 5.6.3 spiegeln grundlegend die bisherigen Erkenntnisse wider: Besonders bei einer kleinen Menge nicht-annotierter Trainingsdaten kann die Self-Training Methode eine gute Verbesserung erzielen. Das „Med“ Profil erzielt bei 15% verwendeten Annotationen erneut die beste relative Verbesserung. Sind jedoch deutlich mehr annotierte Trainingsdaten vorhanden, wird die relative Verbesserung kleiner. Die Partial Labeling Heuristik ist bei der Verwendung von 50% bzw. 100% der annotierten Trainingsdaten ähnlich oder etwas schlechter als ein harter Schwellenwert. Eventuell ist die Fehlertoleranz ab einer gewissen Menge annotierter Daten hoch genug, sodass die Variation der vollständigen Sequenz für das Training wichtiger ist als die Fehlervermeidung. Dies könnte mit einer weiteren Versuchsreihe für eine bessere Wahl der Hyperparameter evaluiert werden.
Im Allgemeinen wird die ExpR_0 auf allen CROHME Testdatensätzen deutlich verbessert. Bei der Verwendung aller Annotationen der Trainingsdaten kann die Erkennungsleistung mithilfe des Self-Trainings und eines harten Schwellenwerts um 7.8 Prozentpunkte gesteigert werden. Auch wenn nur die Hälfte an Annotationen verwendet wird, kann die Erkennungsleistung der 100% Baseline erreicht werden. Die Gewichte des Netzwerks werden also bezüglich des künstlichen „geplanten Einsatzgebietes“ angepasst. Bezogen auf die beiden Forschungsfragen kann die Erkennungsleistung dadurch weiter gesteigert werden oder der benötigte Bedarf an annotierten CROHME Trainingsdaten auf mindestens 50% reduziert werden. Da die Netzwerke dadurch jedoch eine größere Menge an „realen“ Trainingsdaten zur Verfügung haben (wenn auch ohne Annotationen), sind diese nur bedingt mit der Literatur im Folgenden zu vergleichen.

5.7 LITERATURVERGLEICH

Im Bereich der handschriftlichen mathematischen Formelerkennung existieren zwar viele Arbeiten, welche sich mit dem vollständig-überwachten Anlernen von Netzwerken beschäftigen, jedoch nutzen keine dieser Arbeiten semi-überwachte Lernmethoden. Lediglich im Bereich der handgeschriebenen Ziffernerkennung existieren Arbeiten wie [Cec16] oder noch allgemeiner in der Erkennung handgeschriebener Schrift [BCY+21, YLL+22]. Daher verbleibt der Vergleich mit den vollständig-überwachten Ansätzen, welche auf den CROHME Datensätzen evaluiert werden. Dabei sind die Experimente nur bedingt vergleichbar, wenn die Testdaten im Training verwendet werden. Jedoch zeigen diese, dass die Erkennungsrate deutlich über den jetzigen State-of-the-Art gesteigert werden kann, wenn der Grundgedanke der Transduktion auf die Self-Training Methode übertragen wird.

Methode / Modell	CROHME								
	100%			50%			15%		
	14	16	19	14	16	19	14	16	19
DWAP-TD [ZDY+20]	49.1	48.5	51.4						
BTTR [ZGY+21]	55.2	56.6	59.6						
CAN-ABM [LYL+22]	65.9	63.1	64.5						
Baseline	57.0	60.0	63.4	52.5	53.4	59.2	37.1	39.1	40.5
$+\tau = 0.05$	—	—	—	52.6	52.9	59.3	42.6	41.5	43.3
+Syn	56.3	59.5	64.1	54.9	57.0	61.3	43.8	46.6	49.6
$+\tau_M = 0.125$	—	—	—	55.2	56.8	61.8	48.0	50.2	56.0
+Test	61.7	66.2	70.7	61.0	63.0	67.4	51.3	54.8	56.4

Tabelle 5.7.1: *Vergleich der $ExpR_0$ (%) von mehreren Experimenten mit verschiedenen State-of-the-Art Netzwerken auf den CROHME [MZM+19] Datensätzen. Die Experimente erweitern dabei jeweils die Methodik des vorherigen Netzwerks. τ_M bezeichnet den Schwellenwert des Partial Labeling Profils „Mid“.*

In Tabelle 5.7.1 ist die $ExpR_0$ einiger State-of-the-Art Netzwerke für die Erkennung von handschriftlichen mathematischen Formeln abgetragen. Darunter sind die besten Ergebnisse der erweiterten Self-Training Methode abgebildet. Die Experimente bauen dabei in der Tabelle aufeinander auf. Zunächst ist die $ExpR_0$ der Baselines dargestellt, welche nur mit den annotierten Daten vollständig-überwacht angelernt werden. Darauf folgen die Netzwerke, welche mit dem kalibrierten Self-Training und einem festen Schwellenwert $\tau = 0.05$ weiter angelernt werden. Da dafür die restlichen Trainingsdaten genutzt werden, existieren keine Einträge für das mit 100% Trainingsdaten angelernte Netzwerk. Darunter werden die Netzwerke zuvor mit synthetischen Daten vortrainiert und das kalibrierte Self-Training wiederholt. Zudem folgt darauf das erweiterte Self-Training mit der Partial Labeling Heuristik, welche das „Med“ Profil und $\tau_M = 0.125$ nutzt. Abschließend wird die erweiterte Self-Training Methode mit den Testdaten als zusätzliche nicht-annotierte Daten untersucht.
Zu erkennen ist eine deutliche Verbesserung durch die Self-Training Methode. Jedoch erreichen die Netzwerke, welche keinen Zugriff auf alle Annotationen haben, nicht die Erkennungsleistung der 100% Baseline. Nur wenn die Testdaten ohne deren Anno-

tationen im Training benutzt werden, kann die 100% Baseline übertroffen werden. Im Allgemeinen sind die verglichenen vollständig-überwacht angelernten Netzwerke der Literatur besser als die hier vorgestellten Experimente. Dennoch können die beiden zentralen Forschungsfragen nun innerhalb des CROHME Datensatz beantwortet werden: Mithilfe der vorgestellten Self-Training Methode kann die Erkennungsleistung gesteigert, oder der Annotationsaufwand reduziert werden. Sind die Testdaten im Training verfügbar, kann die derzeitige State-of-the-Art Erkennungsleistung deutlich übertroffen werden. Jedoch werden dabei mehr Daten im Trainingsprozess verwendet, wodurch die Erkennungsraten nur bedingt vergleichbar sind. Das RNN-basierte CAN-ABM Netzwerk [LYL+22] erreicht aber deutlich konsistentere und bessere Erkennungsleistungen über die verschiedenen Testdatensätze.

Abschließend lässt sich zusammenfassen, dass die Self-Training Methode die nicht-annotierten Datenpunkte zur Verbesserung der Erkennungsrate nutzt. Die Methode ist dabei effektiver, wenn das Netzwerk nur Zugriff auf wenig annotierte Datenpunkte hat. In diesem Fall ist das synthetische Vortrainieren auch ein wichtiger Bestandteil der Verbesserung der Erkennungsrate. Durch die Kalibrierung wird die Wahl des Schwellenwertes vereinfacht und konsistenter. Gleichzeitig kann die Partial Labeling Heuristik, besonders bei wenigen annotierten Trainingsdaten, zur Verbesserung eingesetzt werden. Dabei sollen Fehler vermieden und die Variation durch möglichst viele Datenpunkte erhöht werden. Die Orakel Experimente haben dies bestätigt, da die Lernmethode eine gewisse Fehlertoleranz hat und durch eine erhöhte Variation eine bessere ExpR_0 erreichen.

5.7.1 *Validierung auf HME100K*

Nachdem die gesamte Methode nun schrittweise aufgebaut und evaluiert wurde, wird diese auf den HME100K Datensatz übertragen und angewandt. Das Ziel ist dabei eine Validierung der vorherigen Ergebnisse auf einem bisher nicht betrachteten Datensatz. Dabei werden die jeweils besten Experimente auf dem CROHME Datensatz auf dem HME100K Datensatz wiederholt. Zudem wird erneut ein synthetisches Vortraining untersucht. Analog zum Literaturvergleich bei CROHME, bauen die untersuchten Experimente aufeinander auf. Zuerst wird auch eine vollständig-überwacht angelernte Baseline betrachtet. Erneut wird diese dann mit einem synthetischen Vortraining erweitert. Danach wird die kalibrierte Self-Training Methode mit der Partial Labeling Heuristik untersucht (Profile „Med“, $\tau_M = 0.25$ bzw. $\tau_M = 0.5$). Schließlich werden erneut die Testdaten als zusätzliche nicht-annotierte Testdaten verwendet.

Methode / Modell	100%	15%	2.5%
DWAP-TD [YLD+22]	61.9		
CAN-DWAP [LYL+22]	67.3		
CAN-ABM [LYL+22]	68.1		
BTTR [YLD+22]	64.1		
SAN [YLD+22]	67.1		
Baseline	64.1	54.9	29.0
+Syn. Baseline	56.5	55.2	32.2
$+\tau_M = 0.25$	—	53.6	32.1
$+\tau_M = 0.5$	—	53.0	35.2
$+\tau_M = 0.25$+Test	65.8	53.2	33.5

Tabelle 5.7.2: *Vergleich der $ExpR_0$ (%) von mehreren Experimenten mit verschiedenen State-of-the-Art Netzwerken auf dem HME100K [YLD+22] Datensatz. Die Experimente erweitern dabei jeweils die Methodik des vorherigen. τ_M bezeichnet den Schwellenwert des Partial Labeling Profils „Mid“.*

Die Erkennungsraten auf dem vollständigen Testdatensatz sind in Tabelle 5.7.2 dargestellt, wobei die Ergebnisse auf den unterschiedlichen Teilmengen der Testdaten im Anhang in Tabelle A.1.7 dargestellt sind. Zu erkennen ist eine identische Erkennungsleistung zwischen dem BTTR Modell und der Baseline des CoMER Modells. Da das CoMER Modell die BTTR Architektur um den Coverage Korrekturterm erweitert, sollte hier der Einfluss von diesem erkennbar sein. Eventuell müssen die Hyperparameter für das Anlernen des Netzwerks angepasst werden, damit das CoMER Modell eine höhere Erkennungsleistung erreicht. Das CoMER Modell übertrifft nur das DWAP Modell und liegt damit hinter allen anderen Ansätzen, die auf den HME100K Datensatz übertragen wurden.

Das synthetische Vortraining hat auf dem HME100K Datensatz für 15% bzw. 2.5% der annotierten Trainingsdaten eine Verbesserung erzielt. Das bekräftigt die Vermutung, dass durch das synthetische Vortraining ein implizites Sprachmodell mittrainiert wird. Falls alle Trainingsdaten verwendet werden, verschlechtert sich die Erkennungsleistung durch das synthetische Vortraining. Eventuell sind genügend Daten vorhanden, sodass der starke visuelle Unterschied der im CROHME-Stil gerenderten LaTeX Sequenzen hinderlich für das weitere Training ist. Die generell gesunkene Komplexität

der Formeln des HME100K Datensatzes, sowie der kleinere Anteil an Hoch- bzw. Tiefstellung sowie griechischen Zeichen (vgl. Abschnitt 5.1.3) könnten weitere Gründe dafür sein. Dadurch könnten die Gewichte der Modellparameter einen Bias hinsichtlich der synthetischen Trainingsdaten haben, welcher durch das weitere vollständig-überwachte Training nicht gänzlich ausgeglichen werden kann. Je weniger annotierte Trainingsdaten vorhanden sind, desto wichtiger scheint ein synthetisches Vortraining zu sein. Sind genügend annotierte Daten vorhanden, kann das implizite Sprachmodell vermutlich durch die realen Trainingsdaten erlernt werden und die visuellen Unterschiede sind hinderlich für das weitere Training.

Die erweiterte Self-Training Methode kann bei dem mit 15% der Trainingsdaten vollständig-überwacht angelernten Netzwerk und unveränderten Hyperparametern keine Verbesserung erzielen. Ein Grund dafür ist die Transformationsliste von RandAug. Durch die Rotation und Translation können Teile der Formel aus dem Eingabebild herausragen. Besonders bei langen Formeln kann dadurch mehr als die Hälfte der Formel fehlen. Wird die modifizierte Transformationsliste aus Tabelle A.1.2 für RandAug verwendet, kann die ExpR_0 auf 55.4% (statt 53.6%, vgl. Tabelle A.1.7) gesteigert werden. Die veränderte Transformationsliste beinhaltet weniger Transformationen, welche zudem schwächer sind. Besonders die Translation wird stark eingeschränkt, damit Formeln nicht aus der Eingabe herausragen. Ein weiterer Grund für den geringen relativen Zugewinn kann der Umfang des Datensatzes sein. Bereits bei 15% der genutzten annotierten Trainingsdaten (11 175) wird der Umfang des gesamten CROHME_{19} Trainingsdatensatzes (8 836) deutlich übertroffen. Unterstützt wird diese Annahme von den Ergebnissen bei 2.5% verwendeten Annotationen. Beide Transformationslisten erzielen eine Verbesserung der ExpR_0. Der relative Unterschied ist hierbei deutlich stärker und vergleichbar mit dem relativen Anstieg der CROHME Experimente (vgl. Tabelle 5.6.2). Mit der veränderten Transformationsliste wird eine Erkennungsrate von 37.1% (statt 35.2%, vgl. Tabelle A.1.7) erreicht, welches einer Verbesserung von 8 Prozentpunkten entspricht.

Eine möglichst große Varianz sowie eine gleichzeitig geringe Fehlerzahl muss zudem durch die Hyperparameter gesteuert werden. Die passende Wahl muss dabei vermutlich in einer weiteren Analyse erarbeitet werden, da sich bei dem geänderten Umfang des Datensatzes auch eine andere Gewichtung der beiden Ziele ergibt (Varianz und Fehleranzahl). Jedoch ist ersichtlich, dass die Trainingsmethode nicht nur auf den CROHME Datensätzen eine Verbesserung erzielen kann, sondern sich in einem gewissen Maße generalisieren lässt. Eine Anpassung der Hyperparameter und der Augmentierungsfunktionen kann dabei unter Umständen nötig sein.

Werden die Testdaten als zusätzliche nicht-annotierte Trainingsdaten verwendet, kann die Erkennungsrate nicht immer gesteigert werden. Eventuell liegt dies erneut an der Wahl der Hyperparameter oder an dem Umfang des Datensatzes. Da im Falle des mit 15% der Annotationen angelernten Netzwerks im Self-Training die restlichen 85% der Trainingsdaten sowie die gesamten Testdaten verwendet werden und der Umfang der restlichen Trainingsdaten den der Testdaten deutlich übersteigt, könnte eine Anpassung auf die Testdaten nur bedingt gelingen. Dies könnte mit einem weiteren Experiment mit einer ausschließlichen Nutzung der Testdaten als nicht-annotierte Datenpunkte untersucht werden. Da das Experiment mit 100% genutzten Trainingsdaten jedoch eine Verbesserung erzielt, scheint eine Anpassung durch das Self-Training möglich zu sein. Werden alle bzw. 2.5% der Annotationen genutzt, wird eine Verbesserung von rund 2 bzw. 4 Prozentpunkten erreicht. Der Zugewinn bleibt jedoch hinter den Ergebnissen auf den CROHME Datensätzen.

Zusammenfassend lässt sich sagen, dass die vorgestellte Self-Training Methode auf beide Datensätze anwendbar ist. Erneut werden dabei bessere Ergebnisse erzielt, wenn weniger Annotationen zur Verfügung stehen. Das synthetische Vortraining hat auf den HME100K Datensatz jedoch nur bedingt eine Verbesserung erzielt. Dies kann an dem unterschiedlichen Bildformat sowie der verringerten Komplexität der HME100K Datensatzes liegen. Werden die Testdaten als weitere nicht-annotierte Trainingsdaten verwendet, kann die Erkennungsleistung auf diesen Daten in manchen Fällen weiter verbessert werden. Das vorherige Zwischenfazit zu den beiden zentralen Forschungsfragen bleibt auch nach den Experimenten auf den HME100K Datensätzen bestehen: Die Erkennungsleistung kann durch nicht-annotierte Daten verbessert oder der Annotationsaufwand verringert werden. Durch den vergleichsweise großen Umfang des HME100K Datensatzes gegenüber dem CROHME Datensatz ist die Methode aber nicht so effektiv, wenn gleichbleibende relative Anteile an Annotationen miteinander verglichen werden.

6 FAZIT

Die Experimente haben gezeigt, dass das Self-Training auch auf die Sequenzerkennung von handschriftlichen mathematischen Formeln übertragbar ist. Verglichen mit den Ergebnissen von FixMatch [SBL+20], scheint die Sequenzerkennung jedoch ein schwierigeres Problem zu sein. Der relative Zugewinn an Erkennungsrate bei einer zunächst ähnlichen Methodik bleibt deutlich hinter den Ergebnissen von FixMatch. Durch die Erweiterung der Methodik kann die relative Verbesserung jedoch gesteigert werden. Die zwei zentralen Forschungsfragen können wie folgt beantwortet werden: Das Self-Training kann den Datenbedarf in einem gewissen Rahmen reduzieren oder aber die Erkennungsleistung mithilfe der nicht-annotierte Datenpunkte verbessern. Je weniger Annotationen dabei vorliegen, desto bessere Ergebnisse können mithilfe des Self-Trainings erzielt werden. Besonders, wenn Daten aus dem Produktionsbetrieb als nicht-annotierte Datenpunkte verwendet werden, kann eine Anpassung auf diese erreicht werden, sodass die Erkennungsrate ohne Annotationsaufwand verbessert werden kann.

Gleichzeitig sind bei der Verwendung von nicht-annotierten Daten einige Dinge zu beachten. Die Experimente haben gezeigt, dass die Hyperparameter passend gewählt werden müssen, um überhaupt eine Verbesserung zu erzielen. Die Kalibrierung erleichtert dies zwar, dennoch müssen viele Entscheidungen getroffen werden. Daher kann eine Datensatz- bzw. problemspezifische Analyse und Suche nach passenden Hyperparametern vermutlich nicht ausbleiben. Darüber hinaus wird bisher immer angenommen, dass das Alphabet der nicht-annotierten Datenpunkte entweder gleich oder eine Teilmenge des Alphabets des Modells ist. Dies kann im Allgemeinen aber nicht garantiert werden. Durch die zwangsläufig falsche Klassifizierung des unbekannten Symbols könnte die allgemeine Erkennungsleistung sinken, da passierende nicht-annotierte Datenpunkte die Gewichte entsprechend der falschen Annotation optimieren. Um dem entgegenzuwirken, müssten vermutlich weiterführende Kalibrierungsmethoden wie die Laplace Approximation [Wal69] genutzt werden, um solche *Out-of-Data* Datenpunkte zu erkennen und aus dem Trainingsprozess zu entfernen.

Zur deutlichen Verringerung der Inferenzdauer können die Pruning Methoden aus [FAO17], sowie das hier vorgestellte Constant Pruning, verwendet werden. Da die Inferenz ein zentraler Bestandteil der Self-Training Methode ist, sollte die Inferenzdauer bei der praktischen Umsetzung beachtet werden. Die Wahl der Hyperparamter der Pruning Methoden sollte jedoch zunächst evaluiert werden, sodass die Erkennungsleistung nicht bzw. nicht deutlich reduziert wird.

Die Kalibrierung des Netzwerks kann für eine einfache und konsistente Wahl des Schwellenwerts der Self-Training Methode genutzt werden. Durch die bessere Verteilung im Konfidenzraum trägt die Kalibrierung zudem zur numerischen Stabilität der Methodik bei. Mithilfe eines kalibrierten Netzwerks kann die Menge an passierenden Pseudolabel einfacher gesteuert werden. Zudem dient der kalibrierte Konfidenzwert als Eingabe der Partial Labeling Heurisitk, welche ebenfalls einfacher mit kalibrierten Konfidenzwerten gesteuert werden kann. Die Experimente zeigen, dass die Kalibrierung eine Verbesserung der Methode erzielt, auch wenn die gleichen Ergebnisse ohne Kalibrierung nachstellbar sein sollten. Dabei hat sich Temperature Scaling sowie LogitNorm als gute Möglichkeit zur deutlichen Verbesserung des erwarteten Kalibrierungsfehlers erwiesen. Bei $\leqslant$ 15% verfügbaren Annotationen erzielt ein mit LogitNorm auf dem CROHME Datensatz trainiertes Netzwerk jedoch eine schlechtere ExpR_0, als der Cross-Entropy Loss. Ab 50% ist die ExpR_0 wiederum höher.

Die Verwendung von synthetischen Daten hat sich als wichtig erwiesen, um vermutlich ein implizites Sprachmodell in einem Vortraining anzulernen. Die größten Verbesserungen werden dabei erreicht, wenn wenig annotierte Datenpunkte vorliegen. Die Experimente auf dem HME100K Datensatz haben jedoch gezeigt, dass das synthetische Vortraining auch hinderlich sein kann. Der Grund dafür könnte in einem unterschiedlichen visuellen Format liegen, oder in einer eher abweichenden Komplexität der Formeln bzw. Verteilung des Alphabets. Trotzdem kann durch das synthetische Vortraining bei 15% bzw. 2.5% verwendeten Annotationen auf dem HME100K Datensatz eine Verbesserung erzielt werden.

Weiterhin hat sich die Partial Labeling Heuristik als eine gute Möglichkeit herausgestellt, um die Variation der Schwellenwert-passierenden Pseudolabel zu erhöhen, während die Fehleranzahl verringert wird. Erneut ist dies besonders bei wenig vorliegen Annotationen wichtig, da die Orakel Experimente gezeigt haben, dass dort eine hohe Varianz sowie eine kleine Fehleranzahl die besten Ergebnisse erzielen. Je mehr Annotationen verwendet werden, desto mehr gleicht die Partial Labeling Heuristik einem harten Schwellenwert. Ab einer gewissen Menge an annotierten Datenpunkten

scheint die Fehlertoleranz genügend groß zu sein, sodass fehlerbehaftete, aber vollständige Sequenzen im Training verwendet werden sollten. Selbiges wird auch bei einer äußerst geringen Menge annotierter Datenpunkte festgestellt, sodass die Partial Labeling Heuristik nur in einem gewissen Bereich besser Ergebnisse erzielt.

Schließlich kann durch die Übertragung des Grundgedanken der Transduktion ein Netzwerk für ein geplantes Einsatzgebiet weiter angepasst werden, ohne dass Annotationen dafür erstellt werden müssen. Mithilfe der erweiterten Self-Training Methode können nicht-annotierte Datenpunkte des Produktionsbetriebs für eine deutliche Verbesserung der ExpR_0 auf den Daten des Produktionsbetriebs genutzt werden. In den Experimenten werden dafür die Testdaten eines Datensatzes als zusätzliche nicht-annotierte Datenpunkte verwendet.

Alles in allem wird also deutlich, dass das Self-Training die nicht-annotierten Datenpunkte nutzen kann und besonders gute Ergebnisse bei wenig verfügbaren Annotationen erreicht. Auf dem CROHME Datensatz kann dadurch die Erkennungsrate eines Netzwerks mit Zugriff auf 15% der Trainingsdaten um den Faktor 1.4 (15.5 Prozentpunkte) gesteigert werden. Die Ergebnisse konnten Teilweise auf dem HME100K Datensatz reproduziert werden. Da dieser jedoch deutlich größer ist, konnte eine vergleichbare Verbesserung nur bei 2.5% genutzten Annotationen erzielt werden. Falls mehr annotierte Datenpunkte der Trainingsdaten verwendet werden, erreicht die Self-Training Methode nur kleine Verbesserungen.

Anhänge

Methode	Intervall	Beschreibung
AutoContrast		Normalisiert den Kontrast, indem die dunkelsten bzw. hellsten Pixel als neues Schwarz bzw. Weiß definiert werden und die Helligkeit der anderen Pixel angepasst wird
Equalize		Ändert die Helligkeit verschiedener Pixelwerte, sodass das Histogramm der Graustufenwerte uniform ist
Invert		Invertiert alle Pixel der Eingabe
Rotate	$\Theta \in [-30, 30]$	Rotiert das Eingabebild um Θ
Posterize	$B \in [1, 4]$	Die Werte in jedem Farbkanal werden auf B Bits reduziert
Solarize	$\tau \in [0, 1]$	Invertiert alle Pixel über einem Threshold τ
SolarizeAdd	$\alpha \in [0, 1]$	Fügt α zu jedem Pixelwert hinzu und führt dann *Solarize* mit $\tau = 0.5$ durch
Color	$\alpha \in [0.1, 1.9]$	Modifiziert die Color-Balance. $\alpha < 1$ reduziert Helligkeit, Kontrast und Saturierung, wohingegen diese bei $\alpha > 1$ erhöht werden
Contrast	$\alpha \in [0.1, 1.9]$	Modifiziert den Kontrast. $\alpha < 1$ verringert den Kontrast, wohingegen $\alpha > 1$ diesen erhöht
Brightness	$\alpha \in [0.1, 1.9]$	Modifiziert die Helligkeit. $\alpha < 1$ verringert die Helligkeit, wohingegen $\alpha > 1$ diese erhöht
Sharpness	$\alpha \in [0.1, 1.9]$	Modifiziert die Schärfe mithilfe einer Faltung. $\alpha < 1$ verringert die Schärfe, wohingegen $\alpha > 1$ diese erhöht
ShearX	$\alpha \in [-0.3, 0.3]$	Schert das Bild mit einer Stärke von α entlang der X-Achse
ShearY	$\alpha \in [-0.3, 0.3]$	Schert das Bild mit einer Stärke von α entlang der Y-Achse
TranslateX	$\alpha \in [-0.3, 0.3]$	Verschiebt das Bild horizontal um $\alpha \cdot \mathbb{W}_x$ Pixel ($\mathbb{W}_x$ ist Breite des Bildes)
TranslateY	$\alpha \in [-0.3, 0.3]$	Verschiebt das Bild vertikal um $\alpha \cdot \mathbb{W}_y$ Pixel ($\mathbb{W}_y$ ist Höhe des Bildes)

Tabelle A.1.1: *Übersicht der verwendeten Transformationen innerhalb der RandAug Methode.*

Methode	Intervall	Beschreibung
AutoContrast		Normalisiert den Kontrast, indem die dunkelsten bzw. hellsten Pixel als neues Schwarz bzw. Weiß definiert werden und die Helligkeit der anderen Pixel angepasst wird
Equalize		Ändert die Helligkeit verschiedener Pixelwerte, sodass das Histogramm der Graustufenwerte uniform ist
Invert		Invertiert alle Pixel der Eingabe
Rotate	$\Theta \in [-30, 30]$	Rotiert das Eingabebild um Θ
SolarizeAdd	$\alpha \in [0, 1]$	Fügt α zu jedem Pixelwert hinzu und führt dann *Solarize* mit $\tau = 0.5$ durch
Brightness	$\alpha \in [0.1, 1.9]$	Modifiziert die Helligkeit. $\alpha < 1$ verringert die Helligkeit, wohingegen $\alpha > 1$ diese erhöht
Sharpness	$\alpha \in [0.1, 1.9]$	Modifiziert die Schärfe mithilfe einer Faltung. $\alpha < 1$ verringert die Schärfe, wohingegen $\alpha > 1$ diese erhöht
ShearX	$\alpha \in [-0.3, 0.3]$	Schert das Bild mit einer Stärke von α entlang der X-Achse
ShearY	$\alpha \in [-0.3, 0.3]$	Schert das Bild mit einer Stärke von α entlang der Y-Achse
TranslateXAbs	$\alpha \in [-10, 10]$	Verschiebt das Bild horizontal um α Pixel
TranslateYAbs	$\alpha \in [-10, 10]$	Verschiebt das Bild vertikal um α Pixel
Scale	$\alpha \in [0.7, 1.4]$	Skaliert das Bild um einen Faktor von α

Tabelle A.1.2: *Übersicht der verwendeten Transformationen innerhalb der modifizierten RandAug Methode.*

τ_{lev}	CROHME$_{14}$ ExpR$_0$					
	50%		35%		15%	
		+Var		+Var		+Var
Baseline	52.5	52.5	48.3	48.3	37.1	37.1
0	52.9	**55.1**	48.9	52.3	42.0	48.5
1	51.8	54.8	48.3	52.2	**42.9**	49.3
2	52.9	52.6	48.8	**53.3**	41.2	48.6
3	**53.2**	52.9	**49.8**	52.9	40.5	**49.9**

Tabelle A.1.3: *Vergleich verschiedener Schwellenwerte im Self-Training mit Orakel Konfidenzmaß und verschiedenen Teilen an annotierten Datenpunkten. Gegenüberstellung mit einer anschließenden zufälligen Variation, bei gleichbleibender absoluter Fehlerzahl und mit den ausschließlich vollständig-überwacht angelernten Netzwerken auf den CROHME$_{14}$ Testdaten.*

τ_{lev}	CROHME$_{16}$ ExpR$_0$					
	50%		35%		15%	
		+Var		+Var		+Var
Baseline	53.4	53.4	51.0	51.0	39.1	39.1
0	**55.4**	**56.9**	**54.3**	55.2	41.6	48.3
1	54.2	56.7	51.1	54.5	42.3	**50.7**
2	54.1	56.1	52.1	55.6	**42.7**	49.4
3	53.4	56.9	51.9	**56.1**	40.9	50.0

Tabelle A.1.4: *Vergleich verschiedener Schwellenwerte im Self-Training mit Orakel Konfidenzmaß und verschiedenen Teilen an annotierten Datenpunkten. Gegenüberstellung mit einer anschließenden zufälligen Variation, bei gleichbleibender absoluter Fehlerzahl und mit den ausschließlich vollständig-überwacht angelernten Netzwerken auf den CROHME$_{16}$ Testdaten.*

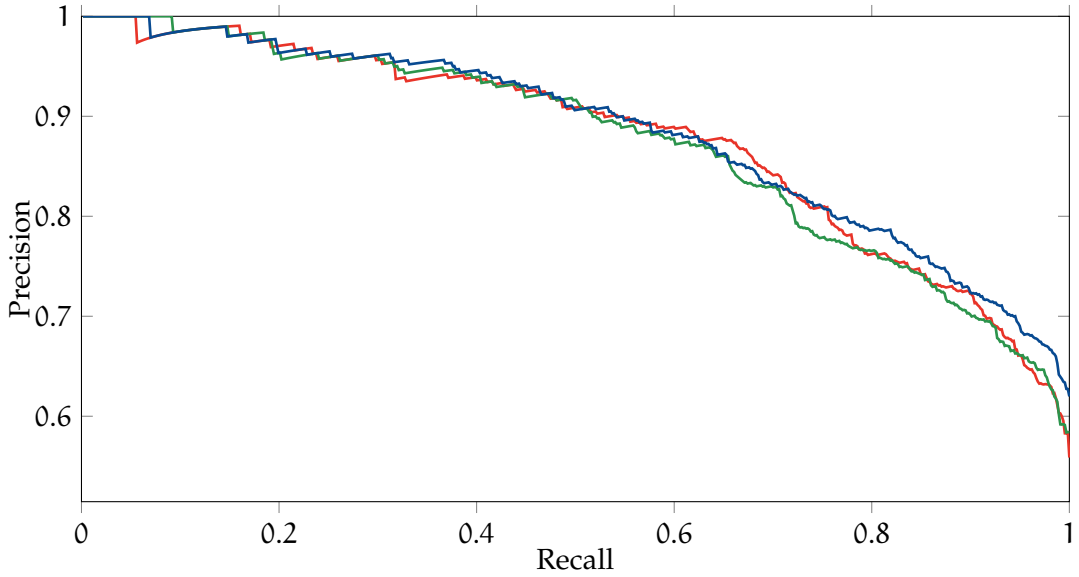


Abbildung A.1.1: *PR-Kurven des ORI, BIMIN und BIMULT Konfidenzmaßes über die CROHME$_{19}$ Testdaten eines mit 35% der Annotationen vollständig-überwacht angelernten Netzwerks.*

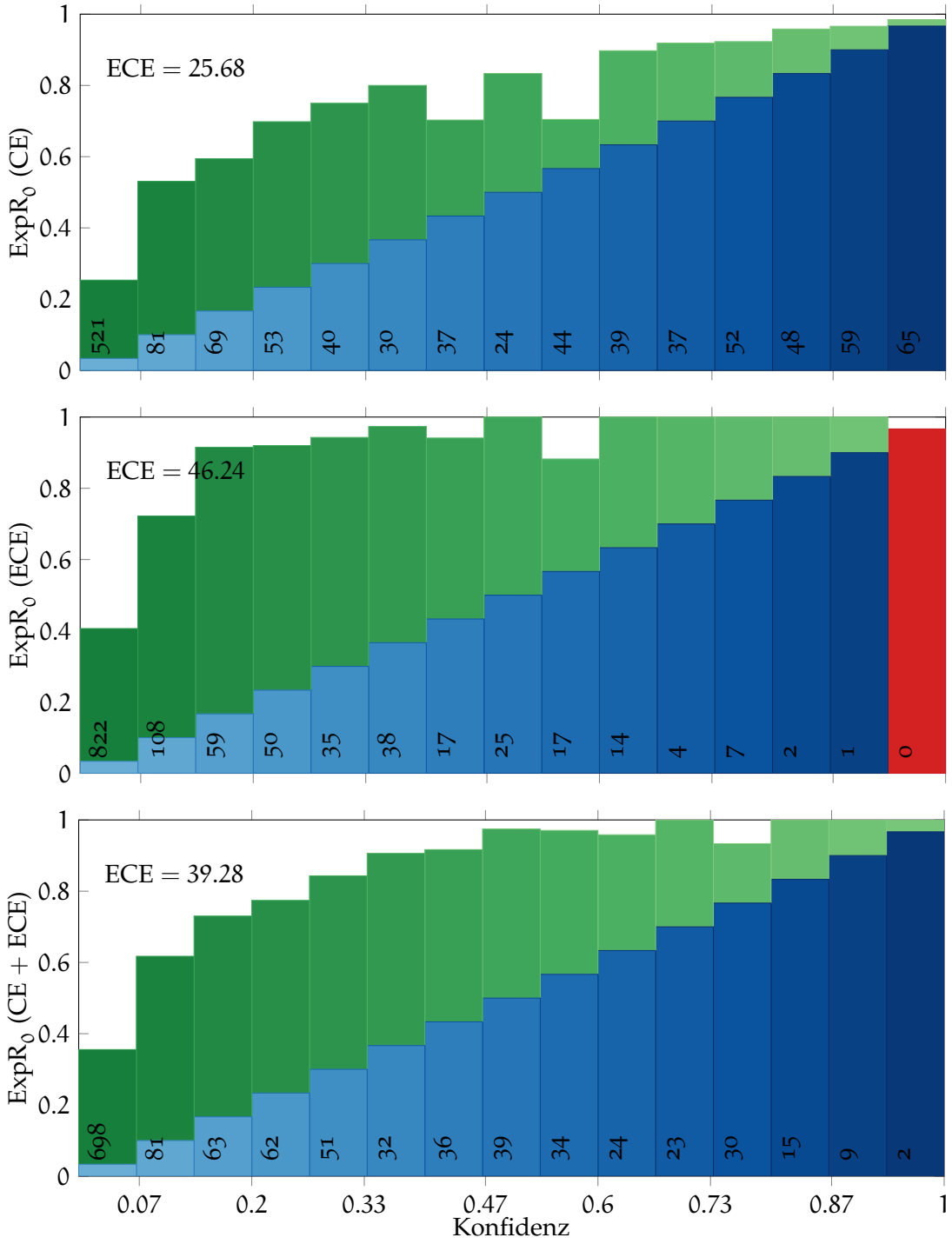


Abbildung A.1.2: *ECE-Diagramme des BIMULT Konfidenzmaßes über die CROHME$_{19}$ Testdaten eines mit 35% der Annotationen vollständig-überwacht angelernten Netzwerks, welches mit Temperature Scaling, verschiedenen Optimierungsmethoden (CE, ECE, CE + ECE) und den CROHME$_{14}$ Testdaten kalibriert wird.*

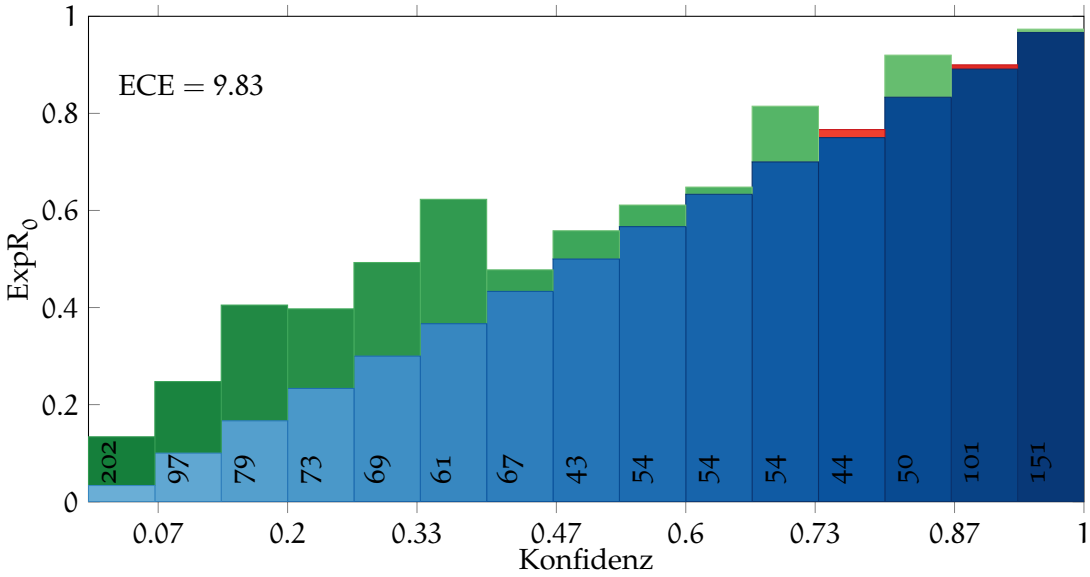


Abbildung A.1.3: *ECE-Diagramm des BIMIN Konfidenzmaßes über die $CROHME_{19}$ Testdaten eines mit 35% der Annotationen vollständig-überwacht angelernten Netzwerks, welches mit Temperature Scaling und den $CROHME_{14}$ Testdaten kalibriert wird.*

Methode	$CROHME_{14}$					
	50%		35%		15%	
	TS	LN + TS	TS	LN + TS	TS	LN + TS
Baseline	**53.3**	**54.1**	48.5	47.6	37.1	37.0
$\tau = 0.95$	52.0	53.7	48.2	47.4	35.4	37.5
$\tau = 0.5$	52.6	53.1	48.5	47.7	39.1	40.7
$\tau = 0.125$	52.0	52.7	48.9	49.2	40.0	42.6
$\tau = 0.05$	52.6	53.5	**49.4**	**50.2**	**42.6**	**43.6**

Tabelle A.1.5: *$ExpR_0$ (%) auf den $CROHME_{14}$ Testdaten mehrerer mit kalibriertem Self-Training, unterschiedlichen Schwellenwerten und verschiedenen Teilen an Annotationen trainierten Netzwerke, verglichen mit den vollständig-überwacht angelernten Baselines. Für die Kalibrierung wird Temperature Scaling sowie LogitNorm mit Temperature Scaling verwendet.*

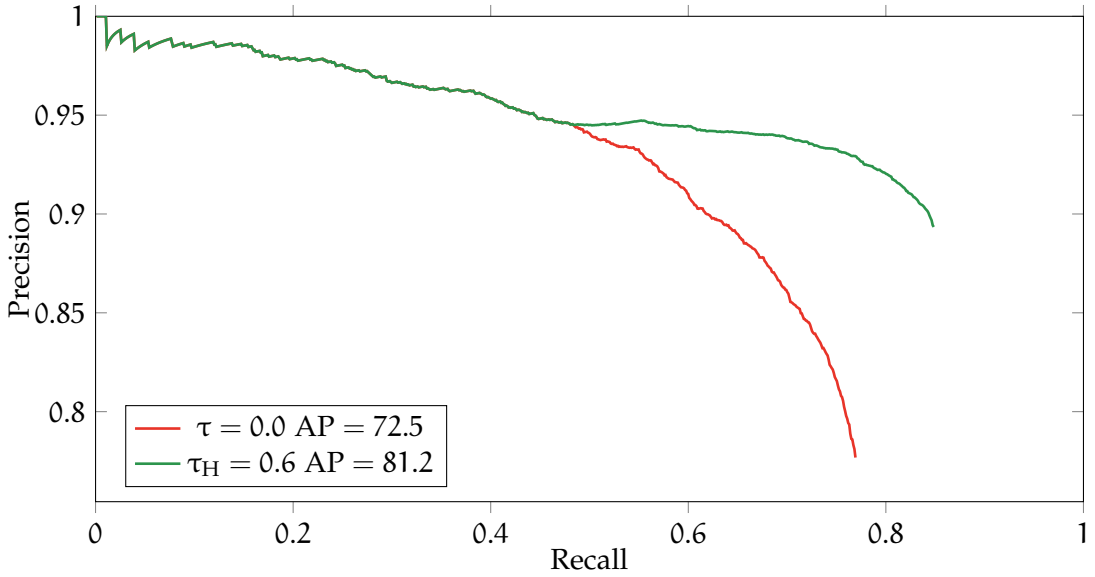


Abbildung A.1.4: *PR-Kurven des BIMIN Konfidenzmaßes mit oder ohne Partial Labeling Heuristik („High" Profil, $\tau_H = 0.6$). Evaluation über die CROHME$_{19}$ Trainingsdaten eines mit 35% der Annotationen vollständig-überwacht angelernten Netzwerks, welches mit Temperature Scaling kalibriert wird. Die Anzahl an True Positives entspricht der Anzahl aller Hypothesen, sodass der Recall von 100% nicht erreicht wird. Dadurch überlagern sich die beiden Graphen für den Vergleich.*

Methode	CROHME$_{16}$					
	50%		35%		15%	
	TS	LN + TS	TS	LN + TS	TS	LN + TS
Baseline	**55.7**	54.1	**52.4**	50.5	39.1	38.6
$\tau = 0.95$	55.1	53.9	51.2	48.7	36.2	36.2
$\tau = 0.5$	53.4	**55.9**	50.5	**51.4**	37.9	39.4
$\tau = 0.125$	53.1	53.4	52.0	50.6	**41.5**	41.9
$\tau = 0.05$	52.9	54.1	52.2	49.9	**41.5**	**43.7**

Tabelle A.1.6: *$ExpR_0$ (%) auf den CROHME$_{16}$ Testdaten mehrerer mit kalibriertem Self-Training, unterschiedlichen Schwellenwerten und verschiedenen Teilen an Annotationen trainierten Netzwerke, verglichen mit den vollständig-überwacht angelernten Baselines. Für die Kalibrierung wird Temperature Scaling sowie LogitNorm mit Temperature Scaling verwendet.*

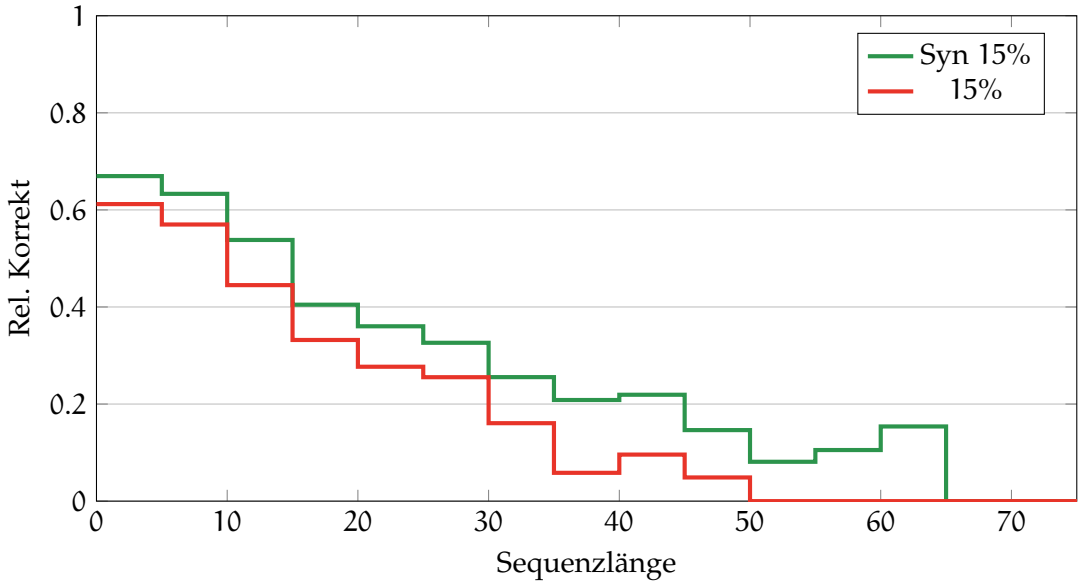


Abbildung A.1.5: *Relative Verteilung der korrekten Vorhersagen über alle CROHME Testdaten, sortiert nach Sequenzlänge in Bins der Breite 5. Vergleich zwischen einem mit 15% Annotationen angelernten Netzwerk mit und ohne synthetisches Vortraining. Wird synthetisches Vortraining verwendet, so werden deutlich längere Sequenzen korrekt vorhergesagt.*

Methode / Modell	100% HME100K				15% HME100K				2.5% HME100K			
	Easy	Med	Hard	Total	Easy	Med	Hard	Total	Easy	Med	Hard	Total
Baseline	76.8	64.7	47.6	64.1	68.6	54.7	38.6	54.9	46.2	26.6	12.0	29.0
+Syn. Baseline	70.9	56.1	39.9	56.5	**69.9**	55.1	37.7	55.2	50.0	30.3	14.0	32.2
$+\tau_M = 0.25$	—	—	—	—	67.7	54.3	35.7	53.6	50.0	31.3	12.0	32.1
$+\tau_M = 0.25$ (T′)	—	—	—	—	69.6	**55.3**	**39.1**	**55.4**	52.0	34.0	16.0	35.0
$+\tau_M = 0.5$	—	—	—	—	67.8	53.4	34.7	53.0	53.8	34.9	13.5	35.2
$+\tau_M = 0.5$ (T′)	—	—	—	—	66.2	53.1	35.8	52.6	**54.2**	**36.9**	**17.3**	**37.1**
$+\tau_M = 0.25$+Test	**77.6**	**66.2**	**51.2**	**65.8**	67.9	53.3	35.2	53.2	51.8	32.9	12.4	33.5

Tabelle A.1.7: *Vergleich der $ExpR_0$ (%) von mehreren Experimenten auf dem HME100K [YLD+22] Datensatz (vgl. Abschnitt 5.1.3). Die Experimente erweitern dabei jeweils die Methodik des vorherigen. Das Self-Training nutzt das „Med“ Profil für die Partial Labeling Heuristik und verschiedene Schwellenwerte. Experimente mit (T′) nutzen die veränderte Transformationsliste aus A.1.2 für die starke Augmentierung mithilfe von RandAug.*

LITERATURVERZEICHNIS

[ASB14] Alvaro, Francisco ; Sanchez, Joan-Andreu ; Benedi, Jose-Miguel: Recognition of on-line handwritten mathematical expressions using 2D stochastic context-free grammars and hidden Markov models. In: *Pattern Recognition Letters* 35 (2014), S. 58–67

[AZG23] Ahlers, Marc ; Zhao, Wenqi ; Gao, Liangcai: CoMER: Modeling Coverage for Transformer-based Handwritten Mathematical Expression Recognition with Semi-Supervised Learning Methods. (2023). https://github.com/revidee/CoMER/

[BAP14] Bachman, Philip ; Alsharif, Ouais ; Precup, Doina: Learning with Pseudo-Ensembles. In: *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*. Montreal, Kanada, 2014, S. 3365–3373

[BCB15] Bahdanau, Dzmitry ; Cho, Kyunghyun ; Bengio, Yoshua: Neural Machine Translation by Jointly Learning to Align and Translate. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. San Diego, CA, USA, 2015

[BCC+20] Berthelot, David ; Carlini, Nicholas ; Cubuk, Ekin D. ; Kurakin, Alex ; Sohn, Kihyuk ; Zhang, Han ; Raffel, Colin: ReMixMatch: Semi-Supervised Learning with Distribution Matching and Augmentation Anchoring. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. Virtuell, 2020

[BCG+19] Berthelot, David ; Carlini, Nicholas ; Goodfellow, Ian ; Oliver, Avital ; Papernot, Nicolas ; Raffel, Colin: MixMatch: A Holistic Approach to Semi-Supervised Learning. In: *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*. Vancouver, Kanada, 2019, S. 5050–5060

[BCY+21] Bhunia, Ayan K. ; Chowdhury, Pinaki N. ; Yang, Yongxin ; Hospedales, Timothy M. ; Xiang, Tao ; Song, Yi-Zhe: Vectorization and Rasterization: Self-Supervised Learning for Sketch and Handwriting. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. Virtuell, 2021, S. 5672–5681

[BQX+22] Bian, Xiaohang ; Qin, Bo ; Xin, Xiaozhe ; Li, Jianwu ; Su, Xuefeng ; Wang, Yanfeng: Handwritten Mathematical Expression Recognition via Attention Aggregation Based Bi-directional Mutual Learning. In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. Virtuell, 2022, S. 113–121

[BSF94] Bengio, Y. ; Simard, P. ; Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. In: *IEEE Computer Society Transactions on Neural Networks* 5 (1994), Nr. 2, S. 157–166

[Caf98] Caflisch, Russel E.: Monte Carlo and quasi-Monte Carlo methods. In: *Acta Numerica* 7 (1998), S. 1–49

[Cec16] Cecotti, Hubert: Active graph based semi-supervised learning using image matching: Application to handwritten digit recognition. In: *Pattern Recognition Letters* 73 (2016), S. 76–82

[CGCB14] Chung, Junyoung ; Gulcehre, Caglar ; Cho, Kyunghyun ; Bengio, Yoshua: Empirical evaluation of gated recurrent neural networks on sequence modeling. In: *Proceedings of the Workshop on Neural Information Processing Systems (NIPS)*, 2014

[CMBB14] Cho, Kyunghyun ; Merrienboer, Bart van ; Bahdanau, Dzmitry ; Bengio, Yoshua: On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. In: *Proceedings of the Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST) at the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar, 2014, S. 103–111

[CSG18] Cohen, Gilad ; Sapiro, Guillermo ; Giryes, Raja: DNN or k-NN: That is the Generalize vs. Memorize Question. In: *CoRR* abs/1805.06822 (2018)

[CST11] Cour, Timothee ; Sapp, Ben ; Taskar, Ben: Learning from partial labels. In: *The Journal of Machine Learning Research* 12 (2011), S. 1501–1536

[CTF+23] Chen, Hao ; Tao, Ran ; Fan, Yue ; Wang, Yidong ; Wang, Jindong ; Schiele, Bernt ; Xie, Xing ; Raj, Bhiksha ; Savvides, Marios: SoftMatch: Addressing the Quantity-Quality Trade-off in Semi-supervised Learning. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. Kigali, Rwanda, 2023

[CY98] Chan, Kam-Fai ; Yeung, Dit-Yan: Elastic structural matching for online handwritten alphanumeric character recognition. In: *Proceedings of the International Conference on Pattern Recognition (ICPR)* Bd. 2. Brisbane, Australien, 1998, S. 1508–1511

[CY00a] Chan, Kam-Fai ; Yeung, Dit-Yan: An efficient syntactic approach to structural analysis of on-line handwritten mathematical expressions. In: *Pattern Recognition* 33 (2000), S. 375–384

[CY00b] Chan, Kam-Fai ; Yeung, Dit-Yan: Mathematical expression recognition: a survey. In: *International Journal on Document Analysis and Recognition (IJDAR)* 3 (2000), Nr. 1, S. 3–15. – ISSN 1433–2833

[CZSL20] Cubuk, Ekin D. ; Zoph, Barret ; Shlens, Jon ; Le, Quoc: RandAugment: Practical Automated Data Augmentation with a Reduced Search Space. In: *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*. Virtuell, 2020, S. 18613–18624

[DKI+21] Daxberger, E. ; Kristiadi, A. ; Immer, A. ; Eschenhagen, R. ; Bauer, M. ; Hennig, P.: Laplace Redux — Effortless Bayesian Deep Learning. In: *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*, 2021, S. 20089–20103

[DSRB14] Dosovitskiy, Alexey ; Springenberg, Jost T. ; Riedmiller, Martin ; Brox, Thomas: Discriminative Unsupervised Feature Learning with Convolutional Neural Networks. In: *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*. Montreal, Kanada, 2014, S. 766–774

[EH20] Engelen, Jesper E. ; Hoos, Holger H.: A survey on semi-supervised learning. In: *Machine Learning* 109 (2020), Nr. 2, S. 373–440

[FAO17] Freitag, Markus ; Al-Onaizan, Yaser: Beam Search Strategies for Neural Machine Translation. In: *Proceedings of the Workshop on Neural Machine Translation (NMT) at the Annual Meeting of the Association for Computational Linguistics (ACL)*. Vancouver, Kanada, 2017, S. 56–60

[Fri23] Frichert, Michael: *Datensynthese für die Erkennung mathematischer Ausdrücke mit tiefen neuronalen Netzen*, TU Dortmund, Bachelorarbeit, 2023

[GBC16] Goodfellow, Ian J. ; Bengio, Yoshua ; Courville, Aaron C.: *Deep Learning*. MIT Press, 2016 (Adaptive computation and machine learning). – ISBN 978–0–262–03561–3

[GH10] Gutmann, Michael ; Hyvärinen, Aapo: Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*. Sardinien, Italien, 2010, S. 297–304

[GPSW17] Guo, Chuan ; Pleiss, Geoff ; Sun, Yu ; Weinberger, Kilian Q.: On Calibration of Modern Neural Networks. In: *Proceedings of the International Conference on Machine Learning (ICML)*. Sydney, Australien, 2017, S. 1321–1330

[Gra12a] Graves, Alex: Sequence Transduction with Recurrent Neural Networks. In: *CoRR* abs/1211.3711 (2012)

[Gra12b] Graves, Alex: *Studies in Computational Intelligence*. Bd. 385: *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer, 2012. – ISBN 978–3–642–24797–2

[HAB19] Hein, Matthias ; Andriushchenko, Maksym ; Bitterwolf, Julian: Why ReLU Networks Yield High-Confidence Predictions Far Away From the Training Data and How to Mitigate the Problem. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA, 2019, S. 41–50

[Ham00] Hammer, Barbara: On the approximation capability of recurrent neural networks. In: *Neurocomputing* 31 (2000), Nr. 1-4, S. 107–123. – ISSN 0925–2312

[Hay98] Haykin, Simon: *Neural Networks: A Comprehensive Foundation*. 2nd. Upper Saddle River, NJ, USA : Prentice Hall PTR, 1998. – ISBN 978–0132733502

[Hor91] Hornik, Kurt: Approximation capabilities of multilayer feedforward networks. In: *Neural Networks* 4 (1991), Nr. 2, S. 251–257. – ISSN 0893–6080

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

[HTB20] He, Fukeng ; Tan, Jun ; Bi, Ning: Handwritten Mathematical Expression Recognition: A Survey. In: *Proceedings of the International Conference on Pattern Recognition and Artificial Intelligence (ICPRAI)* Bd. 12068. Zhongshan, China : Springer, 2020, S. 55–66

[IS15] Ioffe, Sergey ; Szegedy, Christian: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: *Proceedings of the International Conference on Machine Learning (ICML)*. Lille, Frankreich, 2015, S. 448–456

[JH10] *Kapitel* K-Means Clustering. In: Jin, Xin ; Han, Jiawei: *Encyclopedia of Machine Learning*. Boston, MA, USA : Springer, 2010. – ISBN 978–0–387–30768–8, S. 563–564

[KDHN14] Kozielski, Michal ; Doetsch, Patrick ; Hamdani, Mahdi ; Ney, Hermann: Multilingual Off-line Handwriting Recognition in Real-world Images. In: *Proceedings of the International Workshop on Document Analysis Systems (DAS)*. Tours, Frankreich, 2014, S. 121–125

[KHH20a] Kristiadi, A. ; Hein, M. ; Hennig, P.: Being Bayesian, Even Just a Bit, Fixes Overconfidence in ReLU Networks. In: *Proceedings of the International Conference on Machine Learning (ICML)* Bd. 119. Virtuell, 2020, S. 5436–5446

[KHH20b] Kristiadi, Agustinus ; Hein, Matthias ; Hennig, Philipp: Fixing Asymptotic Uncertainty of Bayesian Neural Networks with Infinite ReLU Features. In: *CoRR* abs/2010.02709 (2020)

[Knu12] Knut, Donald E.: The Latin Modern Math (LM Math) font. (2012). https://www.gust.org.pl/projects/e-foundry/lm-math

[Kra91] Kramer, Mark A.: Nonlinear principal component analysis using autoassociative neural networks. In: *AIChE Journal* 37 (1991), Nr. 2, S. 233–243

[KRLP99] Kosmala, A. ; Rigoll, G. ; Lavirotte, S. ; Pottier, L.: On-line handwritten formula recognition using hidden Markov models and context dependent graph grammars. In: *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*. Bengaluru, Indien, 1999, S. 107–110

[KRR+20] Kang, Lei ; Riba, Pau ; Rusiñol, Marçal ; Fornés, Alicia ; Villegas, Mauricio: Pay attention to what you read: Non-recurrent handwritten text-Line recognition. In: *CoRR* abs/2005.13044 (2020)

[KSH12] Krizhevsky, Alex ; Sutskever, Ilya ; Hinton, Geoffrey E.: ImageNet Classification with Deep Convolutional Neural Networks. In: *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*. Lake Tahoe, NV, USA, 2012, S. 1106–1114

[KSZQ19] Khan, Asifullah ; Sohail, Anabia ; Zahoora, Umme ; Qureshi, Aqsa S.: A Survey of the Recent Architectures of Deep Convolutional Neural Networks. In: *CoRR* abs/1901.06032 (2019)

[KTW+20] Khosla, Prannay ; Teterwak, Piotr ; Wang, Chen ; Sarna, Aaron ; Tian, Yonglong ; Isola, Phillip ; Maschinot, Aaron ; Liu, Ce ; Krishnan, Dilip: Supervised Contrastive Learning. In: *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*. Virtuell, 2020, S. 18661–18673

[KW07] Keshari, B. ; Watt, S.: Hybrid Mathematical Symbol Recognition Using Support Vector Machines. In: *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)* Bd. 2. Curitiba, Brasilien, 2007, S. 859–863

[L+13] Lee, Dong-Hyun u. a.: Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In: *Proceedings of the International Conference on Machine Learning (ICML) Workshop*. Atlanta, GA, USA, 2013, S. 896

[LA16] Laine, Samuli ; Aila, Timo: Temporal Ensembling for Semi-Supervised Learning. In: *CoRR* abs/1610.02242 (2016)

[LA17] Laine, Samuli ; Aila, Timo: Temporal Ensembling for Semi-Supervised Learning. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. Toulon, Frankreich, 2017

[LBBH98] LeCun, Y. ; Bottou, L. ; Bengio, Y. ; Haffner, P.: Gradient-Based Learning Applied to Document Recognition. In: *Proceedings of the IEEE* 86 (1998), Nr. 11, S. 2278–2324

[Le20] Le, Anh D.: Recognizing handwritten mathematical expressions via paired dual loss attention network and printed mathematical expressions.

In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshops)*. Seattle, WA, USA, 2020, S. 2413–2418

[Lev66] LEVENSHTEIN, V. I.: Binary Codes Capable of Correcting Deletions, Insertions and Reversals. In: *Soviet Physics Doklady* 10 (1966), S. 707

[LJLZ20] LI, Z. ; JIN, L. ; LAI, S. ; ZHU, Y.: Improving Attention-Based Handwritten Mathematical Expression Recognition with Scale Augmentation and Drop Attention. In: *Proceedings of the International Conference on Frontiers of Handwriting Recognition (ICFHR)*. Virtuell, 2020, S. 175–180

[LKJ16] LY, Fei-Fei ; KARPATHY, Andrej ; JOHNSON, Justin: CS231n: Convolutional Neural Networks for Visual Recognition. (2016). https://cs231n.stanford.edu/

[LNC+11] LE, Quoc V. ; NGIAM, Jiquan ; COATES, Adam ; LAHIRI, Abhik ; PROCHNOW, Bobby ; NG, Andrew Y.: On Optimization Methods for Deep Learning. In: *Proceedings of the International Conference on Machine Learning (ICML)*. Bellevue, WA, USA, 2011 (ICML'11), S. 265–272

[LP98] LAVIROTTE, Stephane ; POTTIER, Loic: Mathematical formula recognition using graph grammar. In: *Proceedings of the Conference on Document Recognition* Bd. 3305. San Jose, CA, USA, 1998, S. 44 – 52

[LPM15] LUONG, Thang ; PHAM, Hieu ; MANNING, Christopher D.: Effective Approaches to Attention-based Neural Machine Translation. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Lisbon, Portugal, 2015, S. 1412–1421

[LXH21] LI, J. ; XIONG, C. ; HOI, S. H.: CoMatch: Semi-supervised Learning with Contrastive Graph Regularization. In: *Proceedings of the IEEE Computer Society International Conference on Computer Vision (ICCV)*. Virtuell, 2021, S. 9455–9464

[LYL+22] LI, Bohan ; YUAN, Ye ; LIANG, Dingkang ; LIU, Xiao ; JI, Zhilong ; BAI, Jinfeng ; LIU, Wenyu ; BAI, Xiang: When Counting Meets HMER: Counting-Aware Network for Handwritten Mathematical Expression Recognition. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Tel Aviv, Israel, 2022, S. 197–214

[MGL+18] Min, Erxue ; Guo, Xifeng ; Liu, Qiang ; Zhang, Gen ; Cui, Jianjing ; Long, Jun: A Survey of Clustering With Deep Learning: From the Perspective of Network Architecture. In: *IEEE Access* 6 (2018), S. 39501–39514

[Mil07] Million, Elizabeth: The Hadamard Product. (2007). https://buzzard.ups.edu/courses/2007spring/projects/million-paper.pdf

[ML10] MacLean, Scott ; Labahn, George: Elastic matching in linear time and constant space. In: *Proceedings of the International Workshop on Document Analysis Systems (DAS)*. Boston, MA, USA, 2010, S. 551–554

[ML14] MacLean, Scott ; Labahn, George: A Bayesian model for recognizing handwritten mathematical expressions. In: *CoRR* abs/1409.5317 (2014)

[MVGZG14] Mouchère, Harold ; Viard-Gaudin, Christian ; Zanibbi, Richard ; Garain, Utpal: Competition on Recognition of On-line Handwritten Mathematical Expressions (CROHME). In: *Proceedings of the International Conference on Frontiers of Handwriting Recognition (ICFHR)*. Kreta, Griechenland, 09 2014, S. 791–796

[MVGZG16] Mouchère, Harold ; Viard-Gaudin, Christian ; Zanibbi, Richard ; Garain, Utpal: Competition on Recognition of On-line Handwritten Mathematical Expressions (CROHME). In: *Proceedings of the International Conference on Frontiers of Handwriting Recognition (ICFHR)*. Shenzhen, China, 2016, S. 607–612

[MZM+19] Mahdavi, Mahshad ; Zanibbi, Richard ; Mouchère, Harold ; Viard-Gaudin, Christian ; Garain, Utpal: Competition on Recognition of Handwritten Mathematical Expressions and Typeset Formula Detection (CROHME+TFD). In: *Proceedings of the International Conference on Frontiers of Handwriting Recognition (ICFHR)*, 2019, S. 1533–1538

[NC08] Nguyen, Nam ; Caruana, Rich: Classification with Partial Labels. In: *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*. Las Vegas, NV, USA, 2008, S. 551–559

[NCH15] Naeini, Mahdi P. ; Cooper, Gregory F. ; Hauskrecht, Milos: Obtaining Well Calibrated Probabilities Using Bayesian Binning. In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. Austin, TX, USA, 2015, S. 2901–2907

[Nie83] NIEMANN, H.: *Klassifikation von Mustern*. Springer, 1983. – ISBN 978–3–642–47517–7

[NMC05] NICULESCU-MIZIL, Alexandru ; CARUANA, Rich: Predicting Good Probabilities with Supervised Learning. In: *Proceedings of the International Conference on Machine Learning (ICML)*. Bonn, Deutschland, 2005, S. 625–632

[OMK21] OTTER, Daniel W. ; MEDINA, Julian R. ; KALITA, Jugal K.: A Survey of the Usages of Deep Learning for Natural Language Processing. In: *IEEE Transactions on Neural Networks and Learning Systems* 32 (2021), Nr. 2, S. 604–624

[P+99] PLATT, John u. a.: Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In: *Advances in large margin classifiers* 10 (1999), Nr. 3, S. 61–74

[PAQ23] PATEL, G. ; ALLEBACH, J. ; QIU, Q.: Seq-UPS: Sequential Uncertainty-aware Pseudo-label Selection for Semi-Supervised Text Recognition. In: *Proceedings of the IEEE Computer Society Winter Conference on Applications of Computer Vision (WACV)*. Waikoloa, HI, USA, 2023, S. 6169–6179

[PDXL21] PHAM, Hieu ; DAI, Zihang ; XIE, Qizhe ; LE, Quoc V.: Meta Pseudo Labels. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. Virtuell, 2021, S. 11557–11568

[Ple17] PLEISS, Geoff: Temperature Scaling. (2017). https://github.com/gpleiss/temperature_scaling

[Pow11] POWERS, David: Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation. In: *Journal of Machine Learning Technologies* 2 (2011), S. 37–63

[RBH+15] RASMUS, Antti ; BERGLUND, Mathias ; HONKALA, Mikko ; VALPOLA, Harri ; RAIKO, Tapani: Semi-supervised Learning with Ladder Networks. In: *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*. Montreal, Kanada, 2015, S. 3546–3554

[RDRS21] RIZVE, Mamshad N. ; DUARTE, Kevin ; RAWAT, Yogesh S. ; SHAH, Mubarak: In Defense of Pseudo-Labeling: An Uncertainty-Aware Pseudo-label Selection Framework for Semi-Supervised Learning. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. Virtuell, 2021

[RHS05] Rosenberg, Chuck ; Hebert, Martial ; Schneiderman, Henry: Semi-Supervised Self-Training of Object Detection Models. In: *Proceedings of the IEEE Computer Society Workshop on Applications of Computer Vision (WACV)*, 2005, S. 29–36

[RHW86] Rumelhart, David E. ; Hinton, Geoffrey E. ; Williams, Ronald J.: Learning representations by back-propagating errors. In: *Nature* 323 (1986), Nr. 6088, S. 533–536

[RS21] Rathore, Prithviraj ; Shrivastava, Shikha: A Review on Recent Advances in Recurrent Neural Networks. In: *Journal of Analysis and Computation (JAC)* 15 (2021)

[SBL+20] Sohn, Kihyuk ; Berthelot, David ; Li, Chun-Liang ; Zhang, Zizhao ; Carlini, Nicholas ; Cubuk, Ekin D. ; Kurakin, Alex ; Zhang, Han ; Raffel, Colin: FixMatch: Simplifying Semi-Supervised Learning with Consistency and Confidence. In: *Proceedings of Conference on Neural Information Processing Systems (NIPS)*. Virtuell, 2020

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

[SJT16] Sajjadi, Mehdi ; Javanmardi, Mehran ; Tasdizen, Tolga: Regularization With Stochastic Transformations and Perturbations for Deep Semi-Supervised Learning. In: *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*. Barcelona, Spanien, 2016, S. 1163–1171

[SK19] Shorten, Connor ; Khoshgoftaar, Taghi M.: A survey on Image Data Augmentation for Deep Learning. In: *Journal of Big Data* 6 (2019), S. 60

[SRN17] Sabir, Ekraam ; Rawls, Stephen ; Natarajan, Prem: Implicit Language Model in LSTM for OCR. In: *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*. Kyoto, Japan, 2017, S. 27–31

[SVL14] Sutskever, Ilya ; Vinyals, Oriol ; Le, Quoc V.: Sequence to Sequence Learning with Neural Networks. In: *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*. Montreal, Kanada, 2014, S. 3104–3112

[TLL+16] Tu, Zhaopeng ; Lu, Zhengdong ; Liu, Yang ; Liu, Xiaohua ; Li, Hang: Modeling Coverage for Neural Machine Translation. In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*. Berlin, Germany, 2016

[TM22] Teslya, Nikolay ; Mohammed, Samah: Deep Learning for Handwriting Text Recognition: Existing Approaches and Challenges. In: *Proceedings of the Conference of Open Innovations Association (FRUCT)*. Helsinki, Finnland, 2022, S. 339–346

[Vap06] Vapnik, Vladimir: *Estimation of dependences based on empirical data*. Springer, 2006. – ISBN 978–0–387–34239–9

[VC74] Vapnik, V. ; Chervonenkis, A.: *Theory of Pattern Recognition [in Russisch]*. Moscow : Nauka, 1974. – (Deutsche Übersetzung: W. Wapnik & A. Tscherwonenkis, *Theorie der Zeichenerkennung*, Akademie–Verlag, Berlin, 1979)

[Vit67] Viterbi, A.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. In: *IEEE Computer Society Transactions on Information Theory* 13 (1967), Nr. 2, S. 260–269

[VSP+17] Vaswani, Ashish ; Shazeer, Noam ; Parmar, Niki ; Uszkoreit, Jakob ; Jones, Llion ; Gomez, Aidan N. ; Kaiser, Ł u. ; Polosukhin, Illia: Attention is All you Need. In: *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*. Long Beach, CA, USA, 2017, S. 5998–6008

[Wal69] Walker, A. M.: On the Asymptotic Behaviour of Posterior Distributions. In: *Journal of the Royal Statistical Society. Series B (Methodological)* 31 (1969), S. 80–88

[WCF+22] Wang, Yidong ; Chen, Hao ; Fan, Yue ; Sun, Wang ; Tao, Ran ; Hou, Wenxin ; Wang, Renjie ; Yang, Linyi ; Zhou, Zhi ; Guo, Lan-Zhe ; Qi, Heli ; Wu, Zhen ; Li, Yu-Feng ; Nakamura, Satoshi ; Ye, Wei ; Savvides, Marios ; Raj, Bhiksha ; Shinozaki, Takahiro ; Schiele, Bernt ; Wang, Jindong ; Xie, Xing ; Zhang, Yue: USB: A Unified Semi-supervised Learning Benchmark for Classification. In: *Proceedings of the Conference on Neural Information Processing Systems (NIPS) Datasets and Benchmarks Track*. Virtuell, 2022

[WCH+23] Wang, Yidong ; Chen, Hao ; Heng, Qiang ; Hou, Wenxin ; Fan, Yue ; ; Wu, Zhen ; Wang, Jindong ; Savvides, Marios ; Shinozaki, Takahiro ; Raj, Bhiksha ; Schiele, Bernt ; Xie, Xing: FreeMatch: Self-adaptive Thresholding for Semi-supervised Learning. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. Kigali, Rwanda, 2023

[WDS+20] Wolf, Thomas ; Debut, Lysandre ; Sanh, Victor ; Chaumond, Julien ; Delangue, Clement ; Moi, Anthony ; Cistac, Pierric ; Rault, Tim ; Louf, Rémi ; Funtowicz, Morgan ; Davison, Joe ; Shleifer, Sam ; Platen, Patrick von ; Ma, Clara ; Jernite, Yacine ; Plu, Julien ; Xu, Canwen ; Scao, Teven L. ; Gugger, Sylvain ; Drame, Mariama ; Lhoest, Quentin ; Rush, Alexander M.: Transformers: State-of-the-Art Natural Language Processing. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Virtuell, 2020, S. 38–45

[WKLQ21] Wang, Xiao ; Kihara, Daisuke ; Luo, Jiebo ; Qi, Guo-Jun: EnAET: A Self-Trained Framework for Semi-Supervised and Supervised Learning With Ensemble Transformations. In: *IEEE Computer Society Transactions on Image Processing* 30 (2021), S. 1639–1647

[WLM+22] Wang, Jianfeng ; Lukasiewicz, Thomas ; Massiceti, Daniela ; Hu, Xiaolin ; Pavlovic, Vladimir ; Neophytou, Alexandros: NP-Match: When Neural Processes meet Semi-Supervised Learning. In: *Proceedings of the International Conference on Machine Learning (ICML)*. Baltimore, MD, USA, 2022, S. 22919–22934

[WMZT22] Wang, Qi ; Ma, Yue ; Zhao, Kun ; Tian, Yingjie: A Comprehensive Survey of Loss Functions in Machine Learning. In: *Annals of Data Science* 9 (2022), Nr. 2, S. 187–212

[WU11] Watt, Stephen ; Underhill, Tom: Ink Markup Language (InkML). (2011). https://www.w3.org/TR/2011/REC-InkML-20110920/

[WXC+22] Wei, Hongxin ; Xie, Renchunzi ; Cheng, Hao ; Feng, Lei ; An, Bo ; Li, Yixuan: Mitigating neural network overconfidence with logit normalization. In: *Proceedings of the International Conference on Machine Learning (ICML)*. Baltimore, MD, USA, 2022, S. 23631–23644

[WYZ+21] Wu, Jin-Wen ; Yin, Fei ; Zhang, Yan-Ming ; Zhang, Xu-Yao ; Liu, Cheng-Lin: Graph-to-Graph: Towards Accurate and Interpretable Online Hand-

written Mathematical Expression Recognition. In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2021, S. 2925–2933

[WZ89] Williams, Ronald J. ; Zipser, David: A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. In: *Neural Computation* 1 (1989), Nr. 2, S. 270–280

[WZ95] *Kapitel* Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity. In: Williams, Ronald J. ; Zipser, David: *Backpropagation: Theory, Architectures, and Applications*. USA : L. Erlbaum Associates Inc., 1995. – ISBN 978–0805812589, S. 433–486

[XDH+20] Xie, Qizhe ; Dai, Zihang ; Hovy, Eduard ; Luong, Minh-Thang ; Le, Quoc V.: Unsupervised Data Augmentation for Consistency Training. In: *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*. Virtuell, 2020

[XSY+21] Xu, Yi ; Shang, Lei ; Ye, Jinxing ; Qian, Qi ; Li, Yu-Feng ; Sun, Baigui ; Li, Hao ; Jin, Rong: Dash: Semi-Supervised Learning with Dynamic Thresholding. In: *Proceedings of the International Conference on Machine Learning (ICML)*. Virtuell, 2021, S. 11525–11536

[XYFP23] Xu, Mingle ; Yoon, Sook ; Fuentes, Alvaro ; Park, Dong S.: A Comprehensive Survey of Image Augmentation Techniques for Deep Learning. In: *Pattern Recognition* 137 (2023)

[YLD+22] Yuan, Ye ; Liu, Xiao ; Dikubab, Wondimu ; Liu, Hui ; Ji, Zhilong ; Wu, Zhongqin ; Bai, Xiang: Syntax-Aware Network for Handwritten Mathematical Expression Recognition. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. New Orleans, LA, USA, 2022, S. 4553–4562

[YLL+22] Yang, Mingkun ; Liao, Minghui ; Lu, Pu ; Wang, Jing ; Zhu, Shenggao ; Luo, Hualin ; Tian, Qi ; Bai, Xiang: Reading and Writing: Discriminative and Generative Modeling for Self-Supervised Text Recognition. In: *Proceedings of the ACM International Conference on Multimedia (ACM MM)*. Lisbon, Portugal, 2022, S. 4214–4223

[ZAK+16] Zanibbi, Richard ; Aizawa, Akiko ; Kohlhase, Michael ; Ounis, Iadh ; Topic, Goran ; Davila, Kenny: NTCIR-12 MathIR Task Overview. In:

Proceedings of the NTCIR Conference on Evaluation of Information Access Technologies (NTCIR). Tokyo, Japan, 2016, S. 299–308

[ZBC02] Zanibbi, R. ; Blostein, D. ; Cordy, J.R.: Recognizing mathematical expressions using tree transformation. In: *IEEE Computer Society Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 24 (2002), Nr. 11, S. 1455–1467

[ZDD18] Zhang, Jian shu ; Du, Jun ; Dai, Lirong: Multi-Scale Attention with Dense Encoder for Handwritten Mathematical Expression Recognition. In: *Proceedings of the International Conference on Pattern Recognition (ICPR)*. Beijing, China, 2018, S. 2245–2250

[ZDY+20] Zhang, Jianshu ; Du, Jun ; Yang, Yongxin ; Song, Yi-Zhe ; Wei, Si ; Dai, Lirong: A Tree-Structured Decoder for Image-to-Markup Generation. In: *Proceedings of the International Conference on Machine Learning (ICML)*. Virtuell, 2020, S. 11076–11085

[ZDZ+17] Zhang, Jianshu ; Du, Jun ; Zhang, Shiliang ; Liu, Dan ; Hu, Yulong ; Hu, Jinshui ; Wei, Si ; Dai, Lirong: Watch, attend and parse: An end-to-end neural network based approach to handwritten mathematical expression recognition. In: *Pattern Recognition* 71 (2017), S. 196–206

[ZE01] Zadrozny, Bianca ; Elkan, Charles: Obtaining Calibrated Probability Estimates from Decision Trees and Naive Bayesian Classifiers. In: *Proceedings of the International Conference on Machine Learning (ICML)*. Williamstown, MA, USA, 2001, S. 609–616

[ZE02] Zadrozny, Bianca ; Elkan, Charles: Transforming Classifier Scores into Accurate Multiclass Probability Estimates. In: *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*. Edmonton, Kanada, 2002 (KDD '02), S. 694–699

[ZG22a] Zhao, Wenqi ; Gao, Liangcai: CoMER: Modeling Coverage for Transformer-Based Handwritten Mathematical Expression Recognition. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Tel Aviv, Israel, 2022, S. 392–408

[ZG22b] Zhao, Wenqi ; Gao, Liangcai: Official implementation for ECCV 2022 paper „CoMER: Modeling Coverage for Transformer-based Handwritten

Mathematical Expression Recognition“. (2022). https://github.com/Green-Wood/CoMER/

[ZGY+21] Zhao, Wenqi ; Gao, Liangcai ; Yan, Zuoyu ; Peng, Shuai ; Du, Lin ; Zhang, Ziyin: Handwritten Mathematical Expression Recognition with Bidirectionally Trained Transformer. In: *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*. Lausanne, Schweiz, 2021, S. 570–584

[ZOKB19] Zhai, Xiaohua ; Oliver, Avital ; Kolesnikov, Alexander ; Beyer, Lucas: S4l: Self-supervised semi-supervised learning. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA, 2019, S. 1476–1485

[ZWH+21] Zhang, Bowen ; Wang, Yidong ; Hou, Wenxin ; Wu, Hao ; Wang, Jindong ; Okumura, Manabu ; Shinozaki, Takahiro: FlexMatch: Boosting Semi-supervised Learning with Curriculum Pseudo Labeling. In: *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*. Virtuell, 2021, S. 18408–18419

[ZYH+22] Zheng, Mingkai ; You, Shan ; Huang, Lang ; Wang, Fei ; Qian, Chen ; Xu, Chang: SimMatch: Semi-Supervised Learning With Similarity Matching. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. New Orleans, LA, USA, 2022, S. 14471–14481