technische universität
dortmund

Attribute CNNs für die Einzelworterkennung

Bachelorarbeit

Maximilian Springenberg
12. August 2019

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 UND MOTIVATION

Die Transkription von gedruckten oder handschriftlichen Texten in das Digitale stellt eine wichtige Aufgabe im Bereich der Bilderkennung dar. In Zeiten der Digitalisierung ist das robuste Erkennen von Texten, unbeeinflusst von variablen Faktoren wie der Dokumentqualität und des Schreibstils, von Notwendigkeit.

In dieser Arbeit wird eine CNN-Architektur, das PHOCNet [SF18], auf das Problem der Einzelworterkennung angewandt. Bei der Einzelworterkennung ist ein Wortabbild gegeben und es wird nach dessen Transkription gesucht. Hierbei wird des weiteren eine Menge von Wortklassen als gegeben vorausgesetzt, die auch als Lexikon bezeichnet wird. Die Schwierigkeit der Einzelworterkennung von Wortabbildern liegt in der Varianz von Handschriften, sowie, bei Datensätzen von historischen Dokumenten, Degradation von Dokumenten und daraus folgend verschlechterte Bildeigenschaften. Zu verschlechterten Bildeigenschaften können in diesem Kontext beispielsweise durchscheinende Tinte einer unteren Seite, Risse oder verwischte Buchstaben gezählt werden.

Ferner wird ein Attributbasierter Ansatz verwendet, bei dem Bildcharakteristika auf einen Attributvektor abgebildete werden. Der verwendete Attributvektor ist ein *Pyramidal Histogramm of Characters* (PHOC) und encodiert Buchstabenvorkommnisse, sowie deren räumliche Verteilung. Das PHOC ist deterministisch für einen String berechenbar. Dadurch erhalten wir dir Möglichkeit Wortabbilder und Transkriptionen in einen gemeinsamen Unterraum zu überführen, in dem bereits durch vergleichsweise einfache Modelle Wortklassen geschätzt werden können.

Um auf Basis des PHOCNet Wortklassen für Wortabbilder zu schätzen wird das Modell PHOC*W*Rec eingeführt. PHOC*W*Rec umfasst das PHOCNet, ein Lexikon und einen Klassifikator, der Repräsentationen aus dem gemeinsamen Unterraum auf Wortklassen abbildet. Dabei wird das PHOCNet verwendet um für ein jedes Wortabbild das PHOC der respektiven Transkription zu schätzen. Das Lexikon besteht aus Wortklassen der Trainings- und Testdaten. Es induziert auf welche Wortklassen PHOC*W*Rec abbilden kann, da die Klassifikatoren in einem gemeinsamen Unterraum die Ähnlichkeit des geschätzten Attributvektors der Wortabbilder zu den Attributvektoren des Lexikons bestimmen. Die Klassifikatoren wählen die für den ähnlichsten Attributvektor des Lexikon respektive Wortklasse als Ausgabe. Während das Modell für die Attributerkennung mit dem PHOCNet fest gewählt ist, werden verschiedene

Klassifikatoren für das Schätzen der Wortklassen betrachtet. Dabei wird eine nächster Nachbar Suche, ein Probabilistisches Modell und eine nächster Nachbar Suche in einem durch CCA ermittelten Unterraum als Klassifikationsansatz untersucht.

Die Struktur dieser Arbeit ergibt sich wie folgt. In Kapitel 2 werden Grundlagen und in Kapitel 3 verwandte Arbeiten auf welchen das Modell PHOCWRec aufbaut vorgestellt. Das Modell PHOCWRec wird in Kapitel 4 vorgestellt. Darauf folgend werden in Kapitel 5 die Experimente erörtert und gedeutet. Ihm Rahmen der Experimente werden neben den Klassifikatoren auch unterschiedliche Konfigurationen der Attributvektoren, sowie verschiedene Augmentierungsverfahren, normalisierte Skalierungen der Eingabebilder und vortrainierte Modelle für das PHOCNet evaluiert. Abschließend wird im Fazit 6 die in dieser Arbeit vorgestellte Methodik eingeschätzt und eine Perspektive auf künftige Arbeiten diskutiert.

2 GRUNDLAGEN

2.1 HANDSCHRIFTERKENNUNG

Die Handschrifterkennung ist ein Problem, das bereits seit Jahrzehnten in der Computervision erforscht wird.

Klassische Ansätze das Problem zu lösen beruhen auf Hidden-Markov-Models (HMM). Diese Ansätze bestehen im Wesentlichen aus fünf Phasen die nacheinander, vergleichbar mit einer Pipeline, abgearbeitet werden und ihre Ausgaben weiterreichen. Diese Phasen umfassen die Text-Lokalisierung, Zeilen-Extraktion, Normalisierung der Zeilen, Merkmals-Extraktion und eine Dekodierung über zwei Modelle [PF09]. Hierbei muss für jeden dieser Schritte ein eigenes Verfahren verwendet werden. Für die Zeilen-Extraktion könnte ein Ansatz verwendet werden, der auf probabilistischer Segmentierung [LZD+08] aufbaut. Verschiedene Preprocessing Verfahren, die etwa die segmentierten Zeilen binarisieren, den segmentierten Bereich auf die Zeilengrundlinie ausrichten und den Text scheren um Kursivschrift zu normaliseren, können für die Normalisierung der Zeilen angewandt werden. Für die Merkmals-Extraktion wird für gewöhnlich ein *sliding window* Ansatz verwendet, wobei ein schmales Fenster entlang der Zeile verschoben wird und weitere temporale Segmentierungen vornimmt. Auf Basis dieser temporalen Segmente können dann Bildmerkmale wie Steigungen und geometrische Eigenschaften extrahiert werden. Die Dekodierung wird durch ein *writing model* und eine *language model* realisiert. Das writing model schätzt elementare Einheiten, im Kontext der Handschrifterkennung sind dies Buchstaben. Das language model schätzt anschließend auf Basis dieser Einheiten beliebige Wortsequenzen. Das HMM wird dabei für das writing model verwendet [PF09].

Moderne Ansätze verwenden *Long Short-Term Memory* (LSTM), erstmals vorgestellt von Sepp Hochreiter in [HS97]. Im Wesentlichen wird ein rekurrentes neuronales Netz (RNN) um Speicherzellen erweitert, welche LSTM realisieren. Die Speicherzellen enthalten Ausgaben einzelner Neuronen des neuronalen Netzes. Sie sind miteinander verbunden und können aufeinander Zugreifen. Eine Speicherzelle besitzt zwei *Gates*, konkret ein Input und ein Output Gate. Das Input Gate kann auf Input anderer Speicherzellen zugreifen um zu entscheiden ob bestimmte Informationen in der respektiven Speicherzelle gespeichert werden sollen. Das Output Gate kann wiederum

auf Basis des Inputs anderer Speicherzellen entscheiden, ob auf bestimmte Attribute der respektiven Speicherzelle zurückgegriffen werden soll. Auf diese Weise kann reguliert werden wann auf welchen Speicher zugegriffen wird. In [DKMJ18] wurde ein LSTM Modell für die Einzelworterkennung vorgestellt, das besonders gut auf heterogenen Datensätzen funktioniert und in [GFGS06] wurde ein Ansatz auf Basis von LSTM mit HMM basierten Ansätzen verglichen. Mittlerweile sind LSTM Modelle Teil des *state of the art* in vielen Bereichen der Sequenzanalyse und somit auch in der Handschrifterkennung.

Ferner wurde in [BNK13] ein Hybrid Ansatz von einem HMM und neuronalen Netzen untersucht. Auch für LSTM wurden bereits Hybrid Modelle, wie in [DKN14] untersucht.

2.2 NEURONALE NETZE

In vergangenen Jahren etablierten sich neuronale Netze unter anderem im Bereich der Computer-Vision. So auch in den Teilbereichen der Wort- und Einzelworterkennung [DKMJ18], [PW16]. Neuronale Netze, insbesondere tiefe neuronale Netze, haben den Vorteil, dass sie die verschiedenen Tasks des Preprocessings, der Merkmals-Extraktion und der Klassifizierung in einem Model vereinen. Im Wesentlichen müssen die gleichen Schritte, die auch in einem klassischen Ansatz angewandt werden in dem Modell des neuronalen Netzes realisiert werden. Während für einen auf HMM basierenden Ansatz die zuvor erwähnten Verfahren aufgestellt und passend modelliert werden müssen, geschieht dies im Rahmen der Optimierung eines neuronalen Netzes implizit.

Im Folgenden werden mit dem MLP und dem CNN zwei Typen von neuronalen Netzen erörtert, die essenziell für die Methodik dieser Arbeit sind.

2.2.1 *Multy Layer Perceptron*

Das Multy Layer Perceptron, kurz MLP, ist ein neuronales Netz, dass vergleichbar zu einem Rechennetz Eingaben durch einen Graph propagiert. Dabei sind die Kanten gewichtet und die Knoten implizieren Neuronen, bzw. Perceptronen, die die gewichteten Eingaben mit einem Bias und einer Aktivierungsfunktion verrechnen um Ausgaben weiterzuleiten. Im folgenden wird die Funktionsweise des Perceptrons, sowie des MLPs im Detail erörtert. Abschließend werden verschiedene Aktivierungsfunktionen und ein Anwendungsbeispiel für ein MLP als Klassifikator diskutiert.

Perceptron

Das Modell des Perceptron wurde erstmals im Jahr 1958 von F. Rosenblatt vorgestellt [Ros58]. Wie zuvor erwähnt, interpretieren wir in einem MLP die Knoten des Rechennetzes als Perceptronen. Ein Perceptron kann in diesem Kontext als Knoten eines Graphen, der um eine Aktivierungsfunktion erweitert wurde, angesehen werden. So hat ein Perceptron eingehende, gewichtete Kanten $w_i, 0 \leqslant i < n, n \in \mathbb{N}$, die respektive Eingaben $x \in \mathbb{R}^n$ gewichten. Ferner wird ein Bias b auf die gewichtete Eingabe addiert, welcher als ein Schwellenwert für die Eingabe gilt. Konkret bedeutet das, dass einem Perceptron eine Aktivierungsfunktion zugeordnet wird, die wie folgt aufgefasst werden kann [Nie18] [LJY19]:

$$f(x) = \phi(w^T x + b) \tag{2.2.1}$$

Hierbei fassen wir die eingehenden gewichteten Kanten im Vektor w zusammen.

Als Aktivierungsfunktion, können verschiedene Funktionen gewählt werden. Die Wahl der passendsten Aktivierungsfunktion ist hierbei von Faktoren wie der Ein- und Ausgabedaten Repräsentation sowie der Netzwerkarchitektur abhängig.

Modell

Das Multi-Layer-Perceptron, kurz MLP, beschreibt ein feedforward Modell [GBC16], dass weitestgehend lineare Schichten verwendet. Im wesentlichen realisiert ein MLP eine Abbildung $f : \mathbb{R}^n \to \mathbb{R}^m$ von Eingabedaten $x \in \mathbb{R}^n$ zu einer Ausgabe $f(x) = y \in \mathbb{R}^m$. Die Abbildung als Solche ist für gewöhnlich recht komplex und wird daher mit $f(x) = f_l(f_{l-1}(\dots(f_1(x))))$ in Kompositionen von mehreren kleineren Funktionen $f_1, \dots, f_l$ unterteilt, dabei sei die Tiefe des MLP durch l definiert. Die einzelnen Funktionen $f_1, \dots, f_l$ können angelernt werden, da neuronale Netze generell differenzierbar sind. Wie genau diese Funktionen angelernt, bzw. das Modell optimiert wird, wird im Kapitel 2.3.2 zur Optimierung neuronaler Netze behandelt. Betrachten wir nun ein MLP als Komposition von Funktionen für Eingabeparameter x, so können wir die Rechenschritte, bzw. die Komposition der Funktionen, mittels eines Graphen definieren. Konkret werden dabei die Knoten als Perceptronen interpretiert und die Kanten werden gewichtet. Kanten seien gerichtet und der Informationsfluss geschieht in Schichten. Dabei bezeichnen wir die erste Schicht f_1 als Eingabe- und die letzte Schicht f_l als Ausgabeschicht. Genauer bedeutet dies hinsichtlich der Konstruktion der Schichten analog zu f, dass eine Eingabe x jede Schicht genau einmal, in der definierten Reihenfolge, von der Eingabe zur Ausgabeschicht durchläuft. Daher auch die Bezeichnung des MLP als feedforward Modell und Netzwerk [GBC16], [Nie18].

Ferner kann das MLP auch als allgemeiner Funktionsapproximator betrachtet werden. Nehmen wir an, ein Perceptron mit einer vergleichsweise simplen Aktivierungsfunktion

$$f(x) = \begin{cases} 1 & , \sum_i^n w_i x_i + b > 0 \\ 0 & , \text{sonst} \end{cases} \tag{2.2.2}$$

sei für ein MLP gegeben [Nie18]. So fungiert dieses Perceptron im Wesentlichen als Hyperebene für boolsche Funktionen. Durch ein einzelnes Perceptron lassen sich bereits die logischen Funktionen AND, OR, NAND, NOR modellieren. Jedoch ist es nicht möglich eine XOR Funktion mittels eines einzelnen Perceptrons zu modellieren, da dafür mehrere Hyperebenen nötig wären. Beruhend auf das zuvor behandelte Prinzip der Schichten in einem MLP, können etwa die Schichten $f_1((x_1, x_2)^T) = (\text{OR}(x_1, x_2), \text{NAND}(x_1, x_2))^T$ und $f_2((x_1, x_2)^T) = (\text{AND}(x_1, x_2))$ für ein MLP definiert werden. Betrachten wir nun, dass die Funktion XOR, von z.B. zwei Eingaben x_1, x_2, durch die Komposition

$$\text{XOR}(x_1, x_2) = \text{AND}(\text{OR}(x_1, x_2), \text{NAND}(x_1, x_2)) = f_2(f_1((x_1, x_2)^T)) = f \tag{2.2.3}$$

hergeleitet werden kann, wird klar, dass durch zwei Perceptronen, die die OR und NAND Funktionen realisieren und deren Ausgaben an ein Perceptron, dass eine AND Funktion realisiert, weitergeleitet werden, auch das XOR modelliert werden kann. Für Komplexere Funktionen ist es also nötig Perceptronen zu kombinieren.

Mittels einer Schicht kann durch eine Hyperebene ein Raum, bzw. eine Menge von Eingaben in zwei Unterräume eingeteilt werden, mittels zwei Schichten kann eine konvexe Mengen separiert werden und mit drei Schichten können mehrere konvexe Mengen separiert, bzw. klassifiziert werden. Insbesondere folgt daraus, dass hinsichtlich der Berechenbarkeit ein MLP nach drei Schichten alle logischen Operation abdecken und somit beliebige Funktionen approximieren kann. Jedoch sind nicht alle Probleme für neuronalen Netzen so einfach, wie das zuvor beschriebene XOR und obwohl es theoretisch möglich wäre neuronale Netze mit nur drei Schichten oder weniger zu modellieren, finden in der Regel tiefe neuronale Netze mehr Anwendung, da sie für komplexe Probleme bessere Approximationen finden können.

Aktivierungsfunktionen

Gegeben sei ein MLP, für das ein paar Gewichte nicht optimal gewählt sind. Dies resultiert in unerwünschten Ausgaben für das Modell. Das zugrundeliegende Problem

ist nun, dass Gewichte angeglichen werden müssen. Mit einer Stufenfunktion als Aktivierungsfunktion ist es jedoch nicht offensichtlich welche Auswirkungen Änderungen von Parametern des MLP auf die Ausgabe haben. Dies liegt an den zu großen, diskreten Sprüngen der Ausgabe eines Perceptrons von 0 auf 1. Im folgenden werden lineare und nicht lineare Aktivierungsfunktionen, sowie ihre Vorteile für neuronale Netze diskutiert.

Eine Möglichkeit das beschriebene Problem zu umgehen ist es die Aktivierungsfunktion so zu definieren, dass sie auf reelle Zahlen abbildet. Dazu eignet sich z.B. die Sigmoid-Funktion

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2.2.4}$$

deren Ausgaben für großes x nahezu 1 und für kleines x nahezu 0 sind. Betrachtet man die Darstellung von σ (Abbildung 2.2.1), so wird ersichtlich, dass die Sigmoid-Funktion eine exponentiell geglättete Funktion von:

$$f(x) = \begin{cases} 1 & , x >= 0 \\ 0 & , \text{sonst} \end{cases} \tag{2.2.5}$$

ist.

In dieser Hinsicht bleibt das Grundprinzip des Perceptrons erhalten und kleinere Änderungen der Gewichte führen zu kleineren Änderungen der Ausgaben. Folglich eignet sich die Sigmoid-Funktion oft besser für das Angleichen von Gewichten, ferner auch für die Optimierung von neuronalen Netzen. Ein weiterer Vorteil ist, dass die Ausgaben eines Sigmoid-Perceptrons, aufgrund ihres Wertebereiches, auch als Wahrscheinlichkeiten interpretiert werden können [Nie18], [GBC16].

Für tiefere neuronales Netze birgt die Sigmoid-Funktion jedoch das Problem des Vanishing-Gradient [Hoc98]. Hierbei wird nach mehrfachem Ableiten der Gradient immer kleiner, und auf Basis des Gradienten können nur noch sehr kleine Veränderungen vorgenommen werden. Der Vanishing-Gradient bewirkt, dass die Optimierung eines neuronalen Netzes stagniert. Wie im Kapitel 2.3.2 zur Optimierung neuronaler Netze angesprochen wird, ist es wichtig, dass Gradienten möglichst aussagekräftig sind, um das Netz zu optimieren. Die Rectified Linear Unit, kurz ReLU (siehe Abbildung 2.2.1) [NH10] ist wie folgt definiert:

$$f_{ReLU}(x) = \begin{cases} x & , x \geqslant 0 \\ 0 & , \text{sonst} \end{cases} \tag{2.2.6}$$

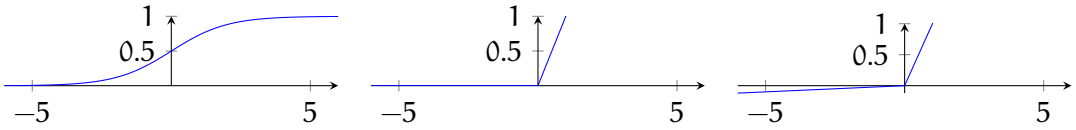

Abbildung 2.2.1: Sigmoid- (links), ReLU- (mitte) [NH10] und leaky ReLU- (rechts) [MHN13], mit $\epsilon = 0,02$, Aktivierungsfunktion.

Sie bietet dabei den Vorteil, dass ihre Ableitung eine Schrittfunktion ist:

$$f'_{ReLU}(x) = \begin{cases} 1 & , x \geqslant 0 \\ 0 & , \text{sonst} \end{cases} \tag{2.2.7}$$

Somit werden Gradienten tieferer Schichten nicht zu klein.

ReLU birgt allerdings auch Nachteile, so können Ausgaben nicht mehr als Wahrscheinlichkeiten interpretiert werden und Gradienten für $x < 0$ werden genullt. Um letzteres zu umgehen wird bei der alternativen Aktivierungsfunktion leaky ReLU [MHN13] ein kleiner Faktor ϵ auf alle Ausgaben kleiner 0 multipliziert:

$$f_{leaky}(x) = \begin{cases} x & , x \geqslant 0 \\ x \cdot \epsilon & , \text{sonst} \end{cases} \tag{2.2.8}$$

Klassifikation

Häufig finden MLPs Anwendung in der *single-* oder *multi label* Klassifikation. Hierbei werden eine oder mehrere Label, bzw. Klassen für eine Eingabe gesucht. Dies kann durch eine binäre Encodierung der für eine Eingabe zutreffenden Klassen in einem Histogramm annotiert werden. Wird mit solchen Embeddings trainiert, so approximiert das resultierende MLP für eine Eingabe dessen Histogramm von Klassen.

Um über das geschätzte Histogramm eine oder mehrere Klassen zu schätzen, können die *Softmax-* oder Sigmoid-Aktivierungsfunktion für die Ausgabeschicht hilfreich sein. Bei der Softmax-Aktivierungsfunktion

$$f_{soft}(\mathbf{x})_i = \frac{e^{x}_i}{\sum_{j=1}^{n} e^{x_j}} \tag{2.2.9}$$

werden Werte auf den Wertebereich $(0, 1]$ abgebildet. Dabei beeinträchtigen hohe Werte des eingehenden Vektors sich gegenseitig. Dementsprechend ist softmax gut für single

label Klassifikation geeignet. Softmax wird deshalb nicht für eine multi label Klassifikation in Betracht gezogen, weil es aufgrund der gegenseitigen Beeinträchtigung der Werte nicht möglich ist, dass mehr als ein Element des Ausgabevektors den Wert 1 annimmt[LJY19] [GBC16].

Für multi label Klassifikation wird deshalb die Sigmoid-Aktivierungsfunktion 2.2.4 verwendet. Diese bildet alle Werte auf den Wertebereich $(0, 1)$ ab, wobei sich die Werte der Eingaben nicht gegenseitig beeinträchtigen. Mit der Sigmoid-Aktivierungsfunktion kann ein MLP konstruiert werden, dass für jede Klasse aus Y die Wahrscheinlichkeit approximiert, dass die Eingabe der respektiven Klasse angehört. Um die Ausgabeschicht eines solchen MLPs mit der Softmax-Aktivierungsfunktion zu vergleichen, kann man den Vergleich zu einer Menge von MLPs mit einer Softmax-Aktivierungsfunktion ziehen, die je nur entgegen einer Klasse aus Y trainiert wurden. In diesem Kontext kann auch das MLP mit der Sigmoid-Aktivierungsfunktion als Komposition mehrerer MLP interpretiert werden, die ihre Gewichte teilen.

2.2.2 *Convolutional Neural Networks*

Neben dem MLP betrachten wir nun das Modell des Convolutional-Neural-Network, kurz CNN. Bei einem CNN werden Daten im Kontext ihrer benachbarten Daten prozessiert. Dies ermöglicht es einem CNN, z.B. für ein Bild, Kanten oder andere Merkmale zu filtern. Ein CNN, dass für einen Klassifizierungstask verwendet wird muss für gewöhnlich mit einem MLP enden.

Dieser Übergang kann geschaffen werden, indem alle Werte der aus Konvolution und Pooling resultierenden Feature-Maps der Ausgabeschicht zu einem Vektor konkateniert werden. Dabei bieten sich für Eingaben variabler Größe Poolingverfahren an, die relativ zur Skalierung der Feature-Maps poolen [HZRS15b], [SRG16]. Das CNN übergibt dem MLP dabei einen Vektor mit encodierten Bildinformationen. Im Folgendem erörtern wir was Konvolution ist und was Konvolution im Kontext eines CNN bedeutet. Abschließend wird mit dem Pooling ein Verfahren gezeigt, dessen Ziel es ist die Invarianz des CNNs gegenüber kleiner Varianz von Eingaben zu verbessern.

Konvolution

Konvolution, oder auch Faltung, kann als ein Verfahren verstanden werden, welches z.B. zeitlichen oder räumlichen Kontext von einer Beobachtung $x(a) \in \mathbb{R}$ für einen Zeitpunkt oder eine Position a berücksichtigt. Beobachtungen zu einem unbekannten Zeitraum werden generell mit 0 gewertet. Dabei werden für den Kontext relevante Beobachtungen mittels einer Funktion $w(a)$ gewichtet [GBC16].

Die Konvolution wird in der allgemeinen Formel

$$s(t) = (x * w)(t) \tag{2.2.10}$$

durch ein $*$ notiert. Im diskreten Fall wird die gewichtete Summe

$$s(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.2.11}$$

für Konvolution betrachtet.

In der Computer Vision werden Elemente des Eingabebildes als die Beobachtungen $x(t)$ interpretiert und ein Kernel $K \in \mathbb{R}^{M \times N}$ realisiert die Gewichtung durch w. Der Kernel ist eine Matrix, die Gewichte enthält. Die Gewichtung erfolgt dadurch, dass der Kernel über das Eingabebild wie ein Fenster geschoben wird und alle Werte in der, durch den Kernel definierten Nachbarschaft, gewichtet und summiert. Dabei kann die Schrittweite, mit der der Kernel über das Bild verschoben wird, variieren und muss definiert werden. Für jede Position wird der Wert einer Ausgabematrix, die wir im Folgenden als *Feature Map* benennen, berechnet, siehe Abbildung 2.2.2. Wie auch beim MLP erschließt sich hierbei ein Graph. Die über den Kernel und der Schrittweite definierten Nachbarschaften haben ausgehende Kannten zu den Gewichten des Kernels mit respektiver Position. Um dies in einer Formel auszudrücken nehmen wir an, ein Eingabebild in schwarz-weiß sei gegeben und die Matrix $I \in [0,1]^{p \times q}$ ordne einer Bildposition ihren Helligkeitswert zu. Um den zum Pixel in Reihe r und Spalte c relevanten Wert für die Feature-Map zu berechnen, kann die Funktion

$$S(r,c) = (I * K)(r,c) = (K * I)(r,c) = \sum_{m=0}^{M} \sum_{n=0}^{N} I(r+m, c+n)K(m,n) \tag{2.2.12}$$

genutzt werden [GBC16].

Unter der Annahme, das ein Eingabebild in schwarz-weiß vorliegt, besteht die Eingabe aus nur einer Matrix. Durch Addieren eines Bias b auf alle Werte der Ausgabe von S und Anwendung einer Aktivierungsfunktion, kann die Feature-Map bestimmt werden. Der Kernel fungiert vergleichbar zur Gewichtsmatrix eines MLP, über die sich zusammen mit dem Bias die Eingabe der Aktivierungsfunktion berechnet.

In einem CNN wird für eine Schicht über die Anzahl von Filtern die Anzahl der ausgegebenen Feature-Maps definiert. Ein Filter besitzt genau so viele Kernel, wie die Eingabe an Feature-Maps besitzt. Nehmen wir an, einer konvolutionären Schicht wird ein schwarz-weiß Bild als Eingabe übergeben, so besitzt jeder Filter genau einen Kernel, da das Eingabebild wie eine Feature-Map betrachtet werden kann [LJY19].

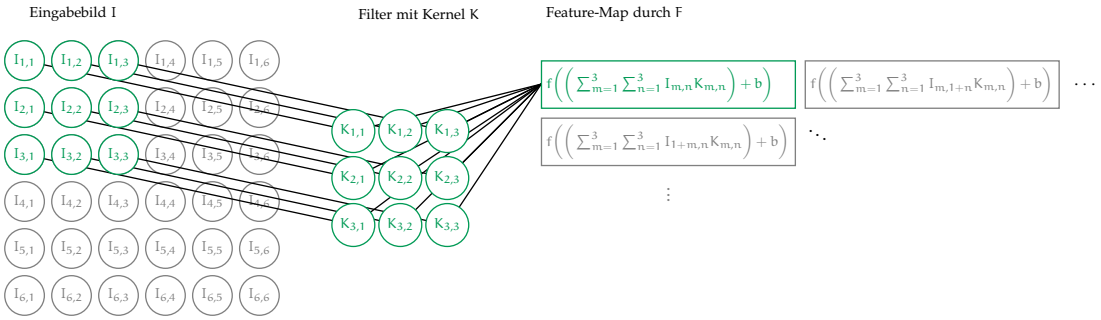

Abbildung 2.2.2: Ein Kernel K wird über das Bild I geschoben. Die Feature-Map wird dabei wie in 2.2.13 definiert, berechnet. Abbildung in Anlehnung an Abbildung 9.1 aus [GBC16].

Erhält eine konvolutionäre Schicht mehrere Feature-Maps als Eingabe, so enthält jeder Filter dieser Schicht respektive Kernel. Eine Feature-Map, bzw. die Ausgabe eines Filters, wird dabei dadurch erhalten, dass S für jede Feature-Map und den respektiven Kernel des Filters angewandt wird. Die Ergebnisse und der Bias werden summiert und der Aktivierungsfunktion übergeben. Für einen Filter mit den Kernels $K_1, \dots, K_d$ einer konvolutionären Schicht mit Bias b, kann, gegeben einer Eingabe I und Aktivierungsfunktion f, die respektive Feature-Map F wie folgt definiert werden:

$$F(r,c) = f\left(\left(\sum_{k=1}^{d} (K_k * I)(r,c)\right) + b\right) \tag{2.2.13}$$

Für ein Eingabebild mit nur einer Matrix für schwarz-weiß Werte, ist in Abbildung 2.2.2 die Berechnung der Feature-Map eines Filters visualisiert.

Pooling

Beim Pooling werden Ausgaben einer Schicht zusammengefasst. Ziel ist es dabei charakteristische Merkmale aufrecht zu erhalten. Dies geschieht, indem Nachbarschaften, bzw. Fenster, mit einer fest definierten Schrittweite auf der Feature-Map auf je einen Wert reduziert werden. Das Pooling-Verfahren wird auf einer jeden Feature-Map angewendet, so erhalten wir, bei einer Schrittweite von 2, einem 2×2 Fenster und k Feature-Maps der Dimensionalität $(40, 40)$, k Ausgaben der Dimensionalität $(20, 20)$. Betrachten wir nun das Max-Pooling (siehe Abbildung 2.2.3). Für jede Position des Fensters wird der maximale Wert in diesem ausgegeben. Durch das Pooling erhofft man sich dabei näherungsweise Invarianz gegenüber der Position und ferner auch

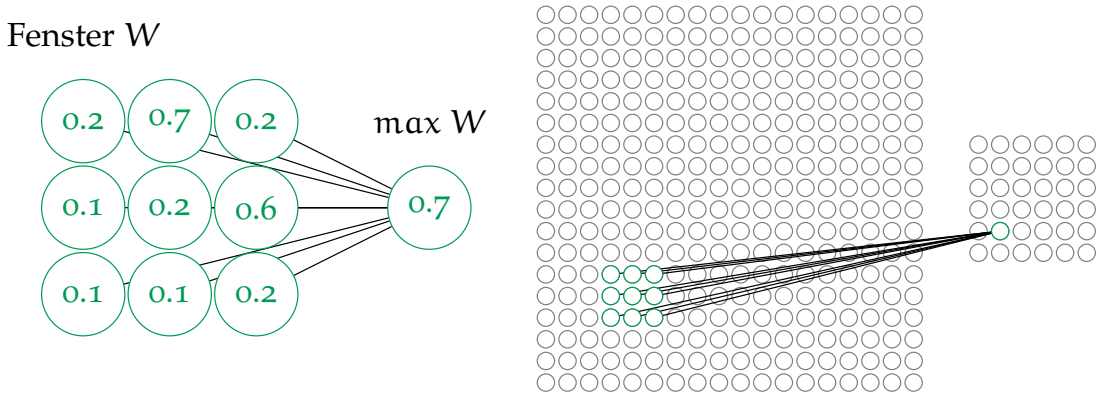

Abbildung 2.2.3: Auf einer Feature-Map wird Max-Pooling angewendet. Hierbei wird ein 3×3 Fenster und eine Schrittweite von 3 verwendet. Die Grün gefärbten Knoten visualisieren eine konkrete Pooling-Operation von einem Fenster. Abbildung in Anlehnung an Kapitel 6 aus [Nie18].

Rotation von Bildmerkmalen. Das Max-Pooling ist am weitesten verbreitet, alternativ können aber auch andere Pooling-Funktionen, wie das Average-Pooling, bei dem der durchschnittliche Wert des Fensters gespeichert wird, das L^2-Norm Pooling, der L^2-Norm der durch das Fenster implizierten Matrix, etc. angewandt werden.

Die Umsetzung von Pooling hat Ähnlichkeiten mit der der Konvolution. In Beiden Fällen wird je ein Ausschnitt der Eingabedaten betrachtet. Wesentliche Unterschiede zeigen sich jedoch in der Berechnung der für die Ausschnitte respektiven Werte, da das Pooling keine Gewichte und auch keinen Bias besitzt. Das Pooling ist eine fest definierte Funktion ohne Parameter des CNNs, die optimiert werden könnten. Bei der Konvolution wird versucht Merkmale, wie z.B. Kanten in einem Bild, durch Gewichtung der Ausschnitte kenntlich zu machen, während beim Pooling versucht wird den Inhalt des Ausschnitts in einem Wert zu encodieren.

Durch Kombination von Konvolution und Pooling ist es also möglich ein Modell zu konfigurieren, dass die Vorteile der Merkmalserkennung der Konvolution und näherungsweise Invarianz gegenüber kleiner Varianz der Eingaben durch das Pooling besitzt. Ferner wird durch das Pooling eine Dimensionsreduktion der Feature-Maps umgesetzt und dadurch die Anzahl von Parametern des CNN reduziert [GBC16], [LJY19].

2.3 OPTIMIERUNG NEURONALER NETZE

Ziel der Optimierung eines neuronalen Netzes ist es eine Zielfunktion zu approximieren. Das Optimierungsverfahren verwendet einen Datensatz und das Optimieren eines Modells auf diesem Datensatz wird oftmals als Training bezeichnet. Die Optimierung wird dadurch realisiert, dass eine Fehlerfunktion, die alle Parameter des neuronalen Netzes berücksichtigt, aufgestellt und mit einem geeigneten Optimierungsverfahren minimiert wird. Über partielles Ableiten dieser Fehlerfunktion nach den Parametern des neuronalen Netzes kann festgestellt werden welche Parameter wie stark angeglichen werden müssen. Wie zuvor angedeutet basiert die Optimierung von neuronalen Netzen darauf, dass das Modell differenzierbar ist. Im folgenden wird die Frage, wie ein neuronales Netz lernt, bzw. wie dessen Fehlerfunktion minimiert und damit das Modell optimiert wird, erörtert.

2.3.1 *Fehlerfunktionen*

Im Kontext neuronaler Netze drückt eine Fehlerfunktion aus, wie stark die geschätzte Ausgabe eines neuronalen Netzes von einer gewünschte Ausgabe abweicht. Wichtig ist hierbei, dass die Fehlerfunktion sämtliche Parameter Θ des neuronalen Netzes berücksichtigt. Dies geschieht implizit, indem eine Ausgabe durch das neuronale Netz berechnet wird. Somit können, durch partielles Ableiten, Gradienten für sämtliche Parameter Θ bestimmt werden. Insbesondere im Unterkapitel zu Stochastic Gradient Descent 2.3.2 ist dies von großer Bedeutung. Für gewöhnlich wird für eine Fehlerfunktion eine Metrik berücksichtigt, die den Ausgabevektor des neuronalen Netzes mit dem Vektor einer optimalen Lösung, oder auch *Embedding* genannt, vergleicht [Nie18], [GBC16].

Möchten wir beispielsweise, gegeben einer Menge von Eingaben X, ein neuronales Netz hinsichtlich der euklidische Distanz seiner Ausgaben $\{f(x)|x \in X\}$ zu den respektiven Richtwerten, bzw. Embeddings, $Y = \{y_0, \ldots, y_n\}$, optimieren, so bietet sich die euklidische Fehlerfunktion:

$$C_{euc}(\Theta) = \frac{1}{2} \sum_{i=1}^{n} \|f(x_i) y_i\|_2^2, \tag{2.3.1}$$

oder der *Mean Squared Error* (MSE):

$$MSE(\Theta) = \frac{1}{n} \sum_{i=1}^{n} (f(x_i) - y_i)^2 \tag{2.3.2}$$

an. Nehmen wir nun an, dass man aus der Menge aller möglichen Lösungen Y_{all} für eine Eingabe x den Vektor aus Y_{all} bestimmen möchte, der $f(x)$ am ähnlichsten ist. Für diesen Ansatz könnte ein neuronales Netz mittels der euklidischen Fehlerfunktion trainiert werden und dementsprechend die euklidische Distanz in einer nächster Nachbar Suche der Ausgabe $f(x)$ in Y_{all} als Ähnlichkeitsmaß verwendet werden. Hinsichtlich Distanzmetriken in hochdimensionalen Räumen hat sich jedoch die Kosinus-Ähnlichkeit, bzw. Kosinusdistanz, $\cos(x, y) = \frac{xy}{||x||_2||y||_2}$ gegenüber der euklidischen Distanz als besser erwiesen [PW16], [SF18]. Möchte man also ein neuronales Netz trainieren und weiß bereits, dass auf dessen Ausgaben eine nächster Nachbar Suche durchgeführt werden soll, bietet es sich an mit der Kosinus-Fehlerfunktion zu trainieren:

$$C_{cos}(\Theta) = \sum_{i}^{n} 1 - \cos(f(x_i), y_i) \tag{2.3.3}$$

Abgesehen vom Fall für $||x||_2 = ||y||_2 = 1$, also normalisierten Eingaben und Embeddings, ist die euklidischen Fehlerfunktion ungleich der Fehlerfunktion für die Kosinus-Ähnlichkeit [SF18]. Deshalb ist es generell sinnvoll neuronale Netze, deren Ausgaben für eine nächster Nachbar Suche verwendet werden mit der Kosinus-Fehlerfunktion zu trainieren.

Eine weitere populäre Fehlerfunktion ist die *Binary Cross Entropy*:

$$BCE(\Theta) = -\frac{1}{n}\sum_{i=1}^{n}(y_i \log(f(x_i)) + (1 - y_i)\log(1 - f(x_i))), \tag{2.3.4}$$

welche insbesondere für Modelle verwendet wird, die Wahrscheinlichkeiten approximieren, da mit $(1 - y_i)$ und $(1 - f(x_i))$ die Gegenwahrscheinlichkeiten von Ausgabe und Embedding berücksichtigt werden.

2.3.2 *Optimierung*

Stochastic Gradient Descent

Ziel von Stochastic Gradient Descent, kurz SGD, ist es die Parameter Θ eines neuronalen Netzes zu optimieren. Dazu wird eine Fehlerfunktion C verwendet, die alle zu optimierenden Parameter berücksichtigt um ein Gütemaß zu bestimmen. Das Anpassen der Gewichte geschieht dabei durch partielles Ableiten von C. Durch die Gradienten ∇C können wir dann die Parameter Θ des Modells schrittweise anpassen.

Im Kontext der Optimierung, bzw. des Trainings, eines neuronalen Netzes mit SGD bedeutet dabei eine Iteration das anpassen der Gewichte:

$$\Theta \leftarrow \Theta - \eta \nabla C \tag{2.3.5}$$

Hierbei ist $\eta \in \mathbb{R}$ ein Wert, der angeben soll wie stark Optimierungsschritte gewichtet sein sollen. Da durch η definiert ist wie stark neue Beobachtungen in den Lernprozess einfließen, nennen wir η auch Learning-Rate [GBC16].

Betrachtet man für ein kleines neuronales Netz, mit nur zwei Parametern, C als eine Funktion, mittels derer eine Ebene aufgespannt wird, so kann man sich SGD als die Talfahrt einer Kugel vorstellen. Abbildung 2.3.1 visualisiert diese Vorstellung. Hierbei sollten wir jedoch in Erwägung ziehen, dass zunächst kein Momentum für SGD definiert ist. So würde sich, in dem Beispiel, die Kugel nicht mehr bewegen, wenn sie ein lokales Minimum erreicht. Zu verstehen ist, dass SGD nicht garantiert ein exaktes globales Minimum für C zu bestimmen. Ferner kann C nicht immer als eine so einfache Funktion, wie in Abbildung 2.3.1 angenommen werden. So besteht die Möglichkeit, dass C mehrere lokale Minima hat. SGD nähert sich also vielmehr einem lokalem Minimum für C an, indem es die Parameter Θ, ausgehend von einer Startkonfiguration, iterativ optimiert.

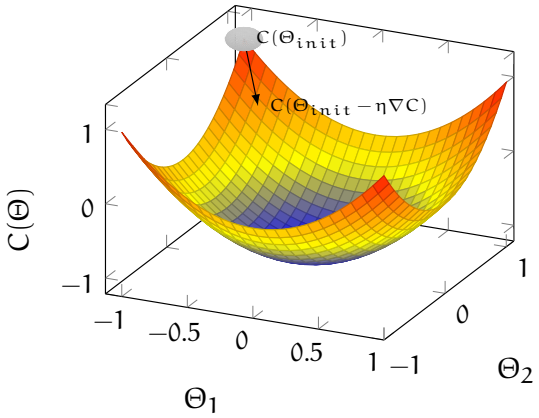

Abbildung 2.3.1: $C((\Theta_1, \Theta_2)^T)$, für zwei Parameter eines neuronalen Netzes. Ziel ist es sich an ein Minimum von C anzunähern. SGD passt dabei Θ_{init} in Abhängigkeit von ∇C und η so an, dass sich Schrittweise einem Minimum angenähert wird. Vergleichbar mit einer Kugel, die ins Tal rollt. Abbildung in Anlehnung an Kapitel 1 aus [Nie18].

Hinsichtlich der Umsetzung von SGD und der Laufzeit der Verfahrens, ist es wichtig zu beachten, dass nach der Definition der Fehlerfunktion alle Daten in Betracht

gezogen werden. Dies würde bedeuten, dass für einen einzigen Schritt 2.3.5 über den gesamten Datensatz X iteriert werden müsste. Laufzeitkosten von $O(|X|)$ für das Bestimmen der Gradienten in einem einzelnen Schritt sind in der Praxis nicht förderlich. Daher werden Mini-Batches, bzw. Teilmengen einer fixen Größe, von X sukzessiv gebildet und die einzelnen Optimierungsschritte auf je einem Mini-Batch ausgeführt. Durch den fixen Wert der Batchgröße, kann die Laufzeit für das Bestimmen der Gradienten, bei fixer Größe der Eingaben, als konstant erachtet werden. Die Wahl der Batchgröße ist hierbei für die Optimierung relevant. Generell approximiert die Varianz eines Mini-Batch, der aus zufällig gewählte Daten besteht, am ehesten die Varianz von Daten im gesamten Datensatz. Ferner ist ein Mini-Batch der beispielsweise nur je einen Eintrag des Datensatzes berücksichtigt für die Optimierung ungeeignet. Bei einer Batchgröße von 1 würden Optimierungsschritte in Form von Gradient Descent für jede Eingabe einzeln erfolgen. Ziel der Optimierung ist es die Fehlerfunktion über alle Daten zu minimieren. Im Falle einer Batchgröße von 1 würde in jedem Optimierungsschritt die Fehlerfunktion über einen Eintrag des Datensatzes minimiert werden. Ferner würde versucht, bei hinreichender Varianz des Datensatzes, in jedem Optimierungsschritt ein anderes Minimum zu approximieren. Deshalb ist es wichtig, den Mini-Batch hinreichend groß zu wählen, um auf Basis von für den Datensatz repräsentativen Teilmengen zu optimieren. Dies erlaubt Training mit mehr Iterationen in kürzerer Zeit [Nie18], [GBC16].

Ferner hat sich hinsichtlich des Optimierens von neuronalen Netzen die Integration des Momentums in der Anpassung der Parameter 2.3.5 etabliert. In der Physik ist Momentum als das Produkt von Masse und Geschwindigkeit eines bewegten Objekts definiert. Für SGD liegt der Fokus aber auf der Geschwindigkeit der Änderungen an den Parametern. So werden Änderungen v_t im folgenden Optimierungsschritt v_{t+1} durch einen Parameter $\mu \in [0, 1)$ für die Gewichtung des Momentums berücksichtigt:

$$v_{t+1} = \mu v_t - \eta \nabla C \tag{2.3.6}$$

$$\Theta' = \Theta + v_{t+1} \tag{2.3.7}$$

Im Kontext des Mini-Batches erhofft man sich durch das Berücksichtigen von vergangenen Optimierungsschritten, dass sich im aktuellen Optimierungsschritt einem lokalen Minimum der Fehlerfunktion über den gesamten Datensatz schneller angenähert wird, aufgrund des sukzessiven Summierens des Momentes. Ein Nachteil des Optimierens mit Moment ist jedoch, dass die Möglichkeit besteht im Optimierungsschritt über ein lokales Minimum hinaus zu schreiten, vergleichbar mit der zuvor angesprochenen Talfahrt einer Kugel, die nicht zwingend aufhört sich zu bewegen, wenn sie einen Tiefpunkt erreicht. Der Hyperparameter μ muss passend gewählt werden. Für gewöhnlich wird ein Werte aus $[0.9, 0.999]$ gewählt oder μ online angepasst [SMDH13].

Backpropagation

Zuvor haben wir durch SGD ein Optimierungsverfahren kennengelernt das auch auf neuronale Netze angewandt werden kann. Jedoch haben wir bis jetzt außer Acht gelassen, wie die Gradienten ∇C berechnet werden. Um die Gradienten eines neuronalen Netzes zu bestimmen wird der Backporpagation-Algorithmus verwendet. Dabei nutzen wir aus, dass wir ein neuronales Netz auch, wie zuvor beschrieben, als Graphen modellieren können.

Die Funktion $f = f_l(f_{l-1}(\dots(f_1)))$ eines neuronalen Netzes besteht aus Kompositionen von Funktionen, die durch je eine Schicht des Netzes modelliert werden. Möchten wir nun die Gradienten aller Schichten bestimmen, so müssen wir partiell Ableiten unter Berücksichtigung der Kettenregel. Die Kettenregel besagt, dass, gegeben zwei Funktionen h, g, mit $p = g(x), q = h(p) = h(g(x))$, wie folgt differenziert werden kann:

$$\frac{dq}{dx} = \frac{dq}{dp}\frac{dp}{dx} \tag{2.3.8}$$

Seien nun h, g Abbildungen der Form $h : \mathbb{R}^m \to \mathbb{R}^n$, $g : \mathbb{R}^n \to \mathbb{R}$, also insbesondere nicht mehr skalar, so können wir wie folgt partiell für einen Parameter $x_i, 0 < i \leqslant |x|$ differenzieren:

$$\frac{\partial q}{\partial x_i} = \sum_{j=1}^{|p|} \frac{\partial q}{\partial p_j}\frac{\partial q}{\partial x_i} \tag{2.3.9}$$

Solche Ausdrücke lassen sich für alle Funktionen $f_1, \dots, f_l$ der respektiven Schichten eines neuronalen Netzes aufstellen. Unter Berücksichtigung der Kettenregel können also alle Gradienten der Schichten und ferner des gesamten Modells bestimmt werden. Hinsichtlich der Laufzeit eines solchen Ansatzes ist es jedoch nicht trivial in welcher Reihenfolge die Teilausdrücke effizient ausgewertet werden und welche Ergebnisse gespeichert werden sollten [Nie18], [GBC16].

Im Unterkapitel 2.2 wurde bereits beschrieben, wie ein neuronales Netz auch als Graph, der ein Rechennetz beschreibt, modelliert werden kann. Für den Backpropagation-Algorithmus iterieren wir dabei rückwärts durch das Netzwerk. Nehmen wir an, wir wollen für ein MLP der Tiefe $L \in \mathbb{N}$ alle Gradienten der Gewichte bestimmen. Ferner seien mit $o_{1,\cdot}, \dots, o_{L,\cdot}$ die Ausgaben aller respektiven Schichten gegeben. Im folgenden verwenden wir die Konvention, dass für eine Matrix A mit $A_{m,\cdot}$ der m-te Reihenvektor adressiert wird. Insbesondere heißt dies, dass wir für jedes Gewicht $w_{l,j}$, der Schicht l ein Fehlersignal $\delta_{l,j}$ bestimmen wollen. Backpropagation beginnt, indem die Fehler-

Signale, bzw. Gradienten, $\varepsilon_{L,\cdot}$ für alle Neuronen der letzten Schicht berechnet werden. Der Gradient des j-ten Neurons der Ausgabeschicht ist dabei wie folgt definiert.

$$\varepsilon_{L,j} = \frac{\delta C}{\delta w_{L,j}} \cdot f'_L(o_{L,\cdot}) \tag{2.3.10}$$

Dieser Fehler Wird dann rückwärts durch das Netzwerk propagiert. Für vorhergehende Schichten sind die Gradienten durch

$$\varepsilon_{l,j} = w^T_{(l+1),\cdot}\varepsilon_{(l+1),\cdot} \cdot f'_l(o_{l,\cdot}) \tag{2.3.11}$$

definiert. Die Gradienten werden durch $w^T_{l+1,\cdot}\varepsilon_{l+1,\cdot}$ also in Gegenrichtung von einem Neuron an alle Neuronen, die adjazent zum respektiven Neuron sind, weitergeleitet und summiert. Ferner ist interessant, dass wir die Aktivierungsfunktion auf Basis der Ausgaben $o_{l,\cdot}$ berechnen (siehe auch Abbildung 2.3.2). Dies erspart uns für jedes Neuron alle Ausgaben bis zur l-ten Schicht erneut zu berechnen. Besonders Vorteilhaft an 2.3.10 und 2.3.11 ist, dass auch der Bias $b_{l,j}$ des j-ten Neuron der l-ten Schicht den selben Gradienten $\varepsilon_{l,j}$ besitzt.

Je nach Optimierungsverfahren können nun die Gewichte und Biase angeglichen werden. Für SGD würden wir z.B. das Gewicht $w_{i,j}$ durch $w_{i,j} \leftarrow w_{i,j}\eta\varepsilon_{i,j}$ anpassen, wobei η die Learning-Rate ist [Nie18], [GBC16].

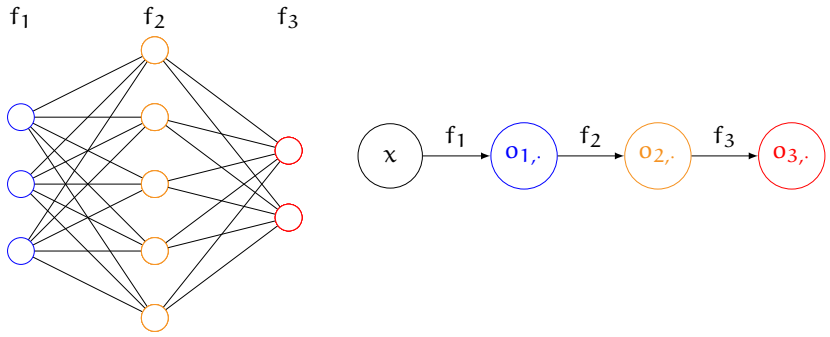

Abbildung 2.3.2: Wir sehen die Forward-Propagation eines MLP visualisert. Erfassen wir die Ausgaben einer jeden Schicht und speichern diese, so erübrigt sich das bestimmen der Ableitungen zu $f'_1(o_{1,\cdot}), f'_2(o_{2,\cdot}), f'_3(o_{3,\cdot})$, da $o_{1,\cdot} = f_1(x), o_{2,\cdot} = f_2(f_1(x)), o_{3,\cdot} = f_3(f_2(f_1(x)))$. Speichern der Ausgaben einer jeden Schicht beschleunigt also das Berechnen der Ableitungen, benötigt jedoch bei tiefen Modellen viel Speicher. Abbildung in Anlehnung an Abbildung 6.9 aus [GBC16].

2.4 CANONICAL CORRELATION ANALYSIS

Canonical Correlation Analysis, kurz CCA, ist ein statistisches Analyseverfahren, dessen Ziel es ist einen Unterraum zu finden, in dem die kanonischen Korrelationen zwischen zwei mehrdimensionalen Datenmengen $X \in \mathbb{R}^{n\times p}, Y \in \mathbb{R}^{n\times q}$ maximal sind. CCA wurde erstmals von H. Hotelling im Jahr 1935 vorgestellt [Hot36], [Hot36]. Dabei werden kanonische Gewichte $w_x \in \mathbb{R}^p, w_y \in \mathbb{R}^q$ ermittelt, die Datenpunkte in einen Unterraum abbilden und ferner auf Basis der kanonischen Korrelationen eine Dimensionsreduktion realisieren können.

Insbesondere gilt hierbei, dass die Dimensionen p, q der Variablen aus X, Y nicht gleich seien müssen. So bietet CCA nicht nur die Möglichkeit einer Dimensionsreduktion von mehrdimensionalen Daten, sondern auch die Möglichkeit Ähnlichkeiten von Daten mit verschiedenen Dimensionen durch Distanzmetriken auszudrücken.

2.4.1 *Allgemeine CCA*

Für ein besseres Verständnis von CCA kann man sich die Datenmengen X, Y als zwei Ansichten x, y auf den selben Datenbestand vorstellen. Das Grundprinz der CCA ist es Korrelationen zwischen diesen Ansichten zu ermitteln und auf Basis derer einen Unterraum zu bestimmen [Hot35], [Hot36], [Bis06]. In diesem Unterraum sollen die Ansichten besser verglichen werden können. Dabei sei $X \in \mathbb{R}^{n\times p}$ die Datenmenge von Ansicht x und $Y \in \mathbb{R}^{n\times q}$ die Datenmenge von Ansicht y. Die Variablen aus X, Y müssen je im Mittel gleich 0 sein und Einheitsvarianz besitzen, damit CCA auf ihnen angewandt werden kann.

Für CCA werden die linearen Transformationen

$$Xw_x = z_x, Yw_y = z_y \tag{2.4.1}$$

betrachtet, wobei $w_x \in \mathbb{R}^p, w_y \in \mathbb{R}^q, z_x \in \mathbb{R}^n, z_y \in \mathbb{R}^n$ gilt. Dabei werden die Ortsvektoren w_x, w_y als die kanonischen Gewichte von X, Y und die Bilder z_x, z_y als die kanonischen Komponenten, oder auch *canonical scores* bezeichnet.

Ziel ist es den Winkel Θ zwischen z_x und z_y zu minimieren. Um den Winkel zu bestimmen eignet sich die Kosinus-Ähnlickeit $\cos(x, y) = \frac{xy}{||x||_2||y||_2}$. Da z_x und z_y Einheitsvektoren sind, reduziert sich die Kosinus-Ähnlichkeit auf das Skalarprodukt von z_x und z_y. Daher können wir $\cos(\Theta)$ auch als die kanonische Korrelation zwischen den Bildern z_x, z_y auffassen. Hinsichtlich einer Transformation von X, Y in einen Unterraum interpretieren wir die Bilder z_x, z_y als die neuen Koordinatenwerte der kanonischen Korrelation $\cos(\Theta)$ im Unterraum [Hot35], [Hot36], [Bis06].

Um CCA zu lösen müssen die kanonischen Gewichte w_x^i, w_y^i und Bilder z_x^i, z_y^i, für die i-te kanonische Korrelation, gefunden werden. Wie zuvor beschrieben wird in der CCA das Paar (z_x, z_y) mit minimalem Winkel, bzw. maximaler kanonischer Korrelation, gesucht. Oftmals möchte man jedoch nicht nur die größte kanonische Korrelation betrachten, sondern eine Anzahl $k \in \mathbb{N} : 1 \leqslant k \leqslant \min\{p, q\}$ von kanonischen Korrelationen. Mittels der kanonischen Gewichte lassen sich dann nach 2.4.1 die respektiven Ansichten x, y in einen Unterraum transformieren [Hot35], [Hot36], [Bis06]. Wenngleich x und y nicht zwingend gleiche Dimensionen besitzen, kann man im weitesten Sinne von einem gemeinsamen Unterraum für die transformierten Ansichten X', Y' sprechen, in dem auch Distanzmetriken als Ähnlichkeitsmaß verwendet werden können.

Lösen von CCA

Eine Möglichkeit CCA zu lösen ist mittels eines Eigenwerteproblems, wie es Hotelling in seiner Veröffentlichung [Hot35], [Hot36] vorstellte. Dazu werden Lagrange Multiplikatoren verwendet um die charakteristische Funktion des Eigenwerte Problems aufzustellen. Zunächst bedingen wir z_x, z_y auf Einheitsvarianz. Aus 2.4.1 folgen also die Bedingungen:

$$z_x^T z_x = w_x^T X^T X w_x = 1 \tag{2.4.2}$$
$$z_y^T z_y = w_y^T Y^T Y w_y = 1 \tag{2.4.3}$$
$$z_x^T z_x = w_x^T X^T Y w_y = 1 \tag{2.4.4}$$

Aus den Definitionen der empirischen Kovarianzmatritzen

$$C_{xx} = \frac{1}{n-1} X^T X \tag{2.4.5}$$
$$C_{yy} = \frac{1}{n-1} Y^T Y \tag{2.4.6}$$

und der Kreuzkovarianz $C_{xy} = \frac{1}{n-1} X^T Y, C_{yx} = C_{xy}^T$ folgt ferner:

$$z_x^T z_x = w_x^T X^T X w_x = w_x^T C_{xx} w_x = 1 \tag{2.4.7}$$
$$z_y^T z_y = w_y^T Y^T Y w_y = w_y^T C_{yy} w_y = 1 \tag{2.4.8}$$
$$z_x^T z_x = w_x^T X^T Y w_y = w_x^T C_{xy} w_y = 1 \tag{2.4.9}$$

Um die Maximale kannonische Korrelation $\cos(\Theta)$ zu erhalten betrachten wir also das Maximierungsproblem:

$$\cos(\Theta) = \max_{z_x, z_y \in \mathbb{R}^n} z_x^T z_y = \max_{w_x \in \mathbb{R}^p, w_y \in \mathbb{R}^1} w_x^T C_{xy} w_y \tag{2.4.10}$$

$$\|z_x\|_2 = 1 \tag{2.4.11}$$

$$\|z_y\|_2 = 1, \tag{2.4.12}$$

wobei die Anwendung der Lagrange Multiplikatoren in der Gleichung:

$$L = w_x^T C_{xy} w_y - \frac{\lambda_1}{2}(w_x^T C_{xx} w_x - 1) - \frac{\lambda_2}{2}(w_y^T C_{yy} w_y - 1) \tag{2.4.13}$$

und partielles Ableiten nach w_x, w_y in:

$$\frac{\delta L}{\delta w_x} = C_{xy} w_y - \lambda_1 C_{xx} w_x = 0 \tag{2.4.14}$$

$$\frac{\delta L}{\delta w_x} = C_{yx} w_x - \lambda_1 C_{yy} w_y = 0 \tag{2.4.15}$$

resultiert. Durch Multipizieren von 2.4.14 mit w_x und 2.4.15 mit w_y erhalten wir $w_c^T C_{xy} w_y - \lambda_1 w_x^T C_{xx} w_x = 0$ und $w_y^T C_{yx} w_x - \lambda_1 w_y^T C_{yy} w_y = 0$. Aus der Einheitsvarianz von z_x und z_y folgt dann $\lambda_1 = \lambda_2 = \lambda$. Lösen wir nun nach w_x auf, so erhalten wir $\frac{C_{xx}^{-1} C_{xy} w_y}{\lambda}$ und ferner die charakteristische Funktion

$$|C_{yy}^{-1} C_{yx} C_{xx}^{-1} C_{xy} - \lambda^2 I| = 0, \tag{2.4.16}$$

wobei I die Einheitsmatrix ist [Hot35], [Hot36], [Bis06].

Lösen wir das Eigenwerteproblem, so erhalten wir durch die für die Eigenwerte $\lambda_1, \ldots, \lambda_{\min\{p,q\}}$, mit $\lambda_1 \geqslant \ldots \geqslant \lambda_{\min\{p,q\}}$, respektiven Eigenvektoren die gesuchten kanonischen Gewichte $w_y^1, \ldots, w_y^{\min\{p,q\}}$. Dabei lässt sich $w_x^1, \ldots, w_x^{\min\{p,q\}}$ aus $w_y^1, \ldots, w_y^{\min\{p,q\}}$ herleiten [Hot35], [Hot36], [Bis06].

2.4.2 *Regularised CCA*

Das Problem der allgemeiner CCA liegt in der charakteristischen Funktion des Eigenwerteproblems $|C_{yy}^{-1} C_{yx} C_{xx}^{-1} C_{xy} - \lambda^2 I| = 0$. C_{yy} und C_{xx} müssen invertierbar und damit insbesondere keine Singulärmatritzen sein. Dies ist leider nicht immer gegeben und gerade im Bereich der Computer Vision wird deshalb oftmals auf Regularised CCA ausgewichen. Bei der regularisierten CCA wird auf der Diagonalen einer jeden

zu invertierenden Matrix eine Konstante $c \in \mathbb{R}$ addiert. Dadurch ist garantiert, dass $C_{yy} + cI$ und $C_{xx} + cI$ keine Singulärmatritzen sind [Vin76].

Bei regularisierter CCA werden die Gleichungen hinsichtlich der Bedingung von z_x, z_y auf Einheitsvarianz

$$z_x^T z_x = w_x^T X^T X w_x = w_x^T (C_{xx} + cI) w_x = 1 \tag{2.4.17}$$

$$z_y^T z_y = w_y^T Y^T Y w_y = w_y^T (C_{yy} + cI) w_y = 1 \tag{2.4.18}$$

und der charakteristischen Funktion, des Eigenwerteproblems,

$$|(C_{yy} + cI)^{-1} C_{yx} (C_{xx} + cI)^{-1} C_{xy} - \lambda^2 I| = 0 \tag{2.4.19}$$

jeweils um die Regularisierung von C_{xx} und C_{yy} erweitert. Die Konstante c muss passend gewählt werden. Da ein optimal gewähltes c nicht ohne weiteres bestimmt werden kann, wird in der Regel ein Validierungsansatz für das Bestimmen von c gewählt.

3 VERWANDTE ARBEITEN

3.1 PHOCNET

S. Sudholt und G. A. Fink stellten 2018 in [SF18] eine CNN-Architektur, das PHOCNet, vor.

Das PHOCNet wurde für das Problem des Word-Spottings, siehe [GSGN17], entwickelt. Beim Word-Spotting ist es das Ziel, Wortvorkommen in einer Sammlung von Dokumenten zu finden. Dazu werden Wortabbilder nach ihrer Ähnlichkeit zum Anfragewort in einer Rückgabeliste sortiert. Hinsichtlich einer Ordnung dieser Rückgabeliste wurde im Modell des PHOCNet die Kosinus-Ähnlichkeit als Metrik verwendet. Dabei wird eine Segmentierung der Wortabbilder benötigt. Ein Vorteil des PHOCNet ist, dass es sowohl *Query by String* (QbS), als auch *Query by Example* (QbE) für das Word-Spotting ermöglicht.

Der Task des PHOCNet ist es für ein Wortabbild einen Attributvektor, das PHOC, zu schätzen. Dieser Attributvektor ist für die Transkription deterministisch geschaffen. Durch das PHOCNet kann eine Abbildung von Wortabbildern und Strings in einen gemeinsamen Vektorraum realisiert werden. Auf Basis der geschätzten Kodierungen können dann, wie auch für die Ordnung der Rückgabeliste, durch Distanzmetriken Ähnlichkeiten von Bildern untereinander oder Bildern und PHOC Encodierungen bestimmt werden.

3.1.1 *Pyramidal Histogram of Characters*

Das PHOCNet überführt Bilder in geschätzte Attributsvektoren. Ein jeder geschätzter Attributvektor soll ein *Pyramidal Histogramm Of Characters* (PHOC) [AGFV14] approximieren. Dabei wird versucht, das PHOC der Transkription des Wortabbildes zu approximieren.

Ein PHOC ist eine Encodierung von einem Wort, bei der Vorkommnisse von Buchstaben binär in einem Histogramm encodiert werden. Ein Teilstring kann als ein *Histogramm Of Characters* (HOC) encodiert werden. Das HOC enthält binäre Einträge für Buchstaben, die im respektiven Teilstring vorkommen, bzw. dessen Attribute. Das PHOC enthält mehrere HOC konkateniert, wobei getreu einer pyramidialen Struktur

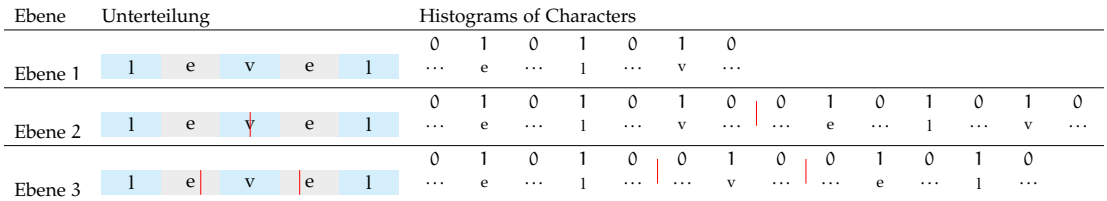

Abbildung 3.1.1: Der Aufbau der Ebenen eines PHOC visualisiert an dem Wort *level*. Die Buchstabenregionen wurden verschiedenfarbig gekennzeichnet. Es ist zu beobachten, wie durch tiefere Ebenen die räumliche Verteilung der Buchstaben im Wort encodiert werden können. Abbildung in Anlehnung an Abbildung 2 aus [SF18].

Wortabschnitte encodiert werden, sodass eine Menge an HOC für Teilwörter bestimmt werden. Die Anzahl der Wortabschnitte einer Ebene nehmen linear zu. So existiert auf der ersten Ebene der pyramidialen Struktur nur ein Abschnitt, auf der zweiten zwei Abschnitte und so fort. Wortabschnitte einer Ebene besitzen gleiche Länge, leeren Schnitt und decken das gesamte Wort ab. Ein Buchstabe gehört einem Wortabschnitt genau dann an, wenn dieser mindestens 50% der Region des Buchstaben belegt. Die Zuordnung eines Buchstaben zu einer Region sei dabei durch die normalisierte Belegung

$$\mathrm{OCC}(k, n) = \left[\frac{k}{n}, \frac{k+1}{n}\right] \tag{3.1.1}$$

definiert [AGFV14]. Hierbei ist n gleich der Länge der Transkription und k bezeichnet den Index des betrachteten Buchstaben. Betrachten wir also wie in Abbildung 3.1.1 die Transkription *level*, so folgt aus der Definition der Buchstabenregionen 3.1.1 insbesondere, dass der Buchstabe *v* auf der zweiten Ebene in beiden Teilwörtern vorkommt. Durch das konkatenieren der aus den Wortabschnitten resultierenden HOC erhält man mit dem resultierenden PHOC eine Kodierung, die die räumliche Verteilung von Buchstaben im String berücksichtigt.

3.1.2 *Architektur*

Die Architektur des PHOCNet bedient sich konvolutionärer Schichten, um Bildmerkmale zu filtern und vollständig verbundener Schichten, um die Merkmale in einen Attributvektor im respektiven PHOC-Format zu überführen. Der Übergang von konvolutionären zu vollständig verbundenen Schichten wird dabei mittels einer *Spatial Pyramid Pooling* (SPP) Schicht mit Max-Pooling geschaffen [HZRS15b]. Diese Schicht

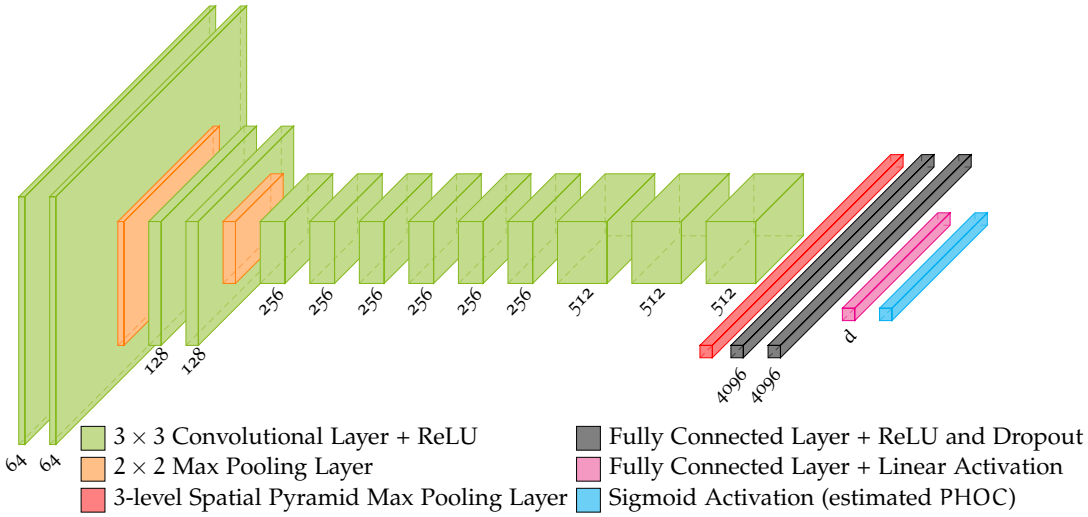

Abbildung 3.1.2: PHOCNet Architektur wie erstmals in der Veröffentlichung [SF18] vorgestellt, für PHOC Repräsentationen der Länge d. Der Übergang von konvolutionären Schichten zum MLP erfolgt über eine SPP-Schicht. Abbildung 3 aus [SF18].

ermöglicht, dass Bilder beliebiger Skalierung von dem neuronalem Netz (NN) ohne Strecken oder Ausschneiden von Bildbereichen prozessiert werden können [SF18].

Alternativ zur SPP Schicht kann auch eine *Temporal Pyramidal Pooling*(TPP) Schicht verwendet werden. Bei dem TPP wird temporal, also entlang von Spalten, gepoolt. Man erhofft sich von der TPP-Schicht, hinsichtlich des Aufbaus eines PHOC, gegenüber einer SPP-Schicht, dass für das PHOC relevante Informationen einfacher extrahiert werden können, bzw. eine TPP Schicht besser angelernt werden kann. Diese Annahme basiert darauf, dass für Buchstaben in Wortabbildern angenommen wird, dass Vertikale Bildabschnitte von größerer Bedeutung sind, als Horizontale. Ferner gleicht der Aufbau der TPP-Schicht den Ebenen eines PHOC. Wie in 3.1.3 visualisiert können Repräsentationen mit niedriger Dimensionalität und mehr Vertikalen Bildabschnitte durch eine TPP-Schicht mit mehr Ebenen realisiert werden, als durch eine SPP-Schicht. In [RSS+18] wurde das Training einer PHOCNet-Architektur bezüglich Augmentierung der Trainingsdaten, Skalierung von Eingabebildern, sowie auch einer TPP-Schicht und *Zoning* [SRG16] untersucht. Dabei konnte jedoch kein signifikanter Unterschied zwischen dem Ansatz des Zonings und einer TPP Schicht festgestellt werden.

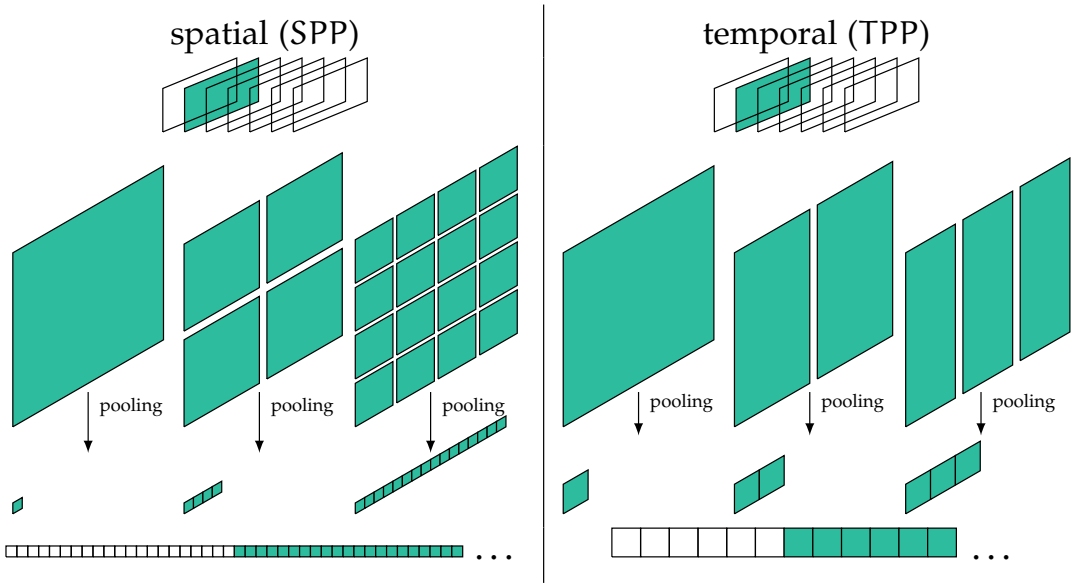

Abbildung 3.1.3: SPP und TPP Schichten mit einer Tiefe von 3 für eine Feature-Map visualisiert. Das, in Abhängigkeit der Schichten, exponentielle Wachstum des SPP resultiert in deutlich größeren Repräsentationen je Feature-Map.

3.2 CNN-N-GRAM

In [PW16] wurde von A. Poznanski und L. Wolf das Model CNN-n-Gram für das Problem der Einzelworterkennung von handgeschriebenem Text vorgestellt. Wie auch das PHOCNet verfolgt CNN-n-Gram einen segmentierungsbasierten Ansatz. In ihrem Verfahren wurde ein CNN benutzt um die n-Gram Encodierungen von Bildtranskriptionen zu approximieren. Wie beim PHOCNet werden also Attributvektoren für Wortabbilder geschätzt. Unterschied hierbei ist jedoch, dass statt einem MLP, welches die aus einem CNN gewonnenen Informationen auf eine Attributkodierung abbildet, 19 MLPs parallel je ein n-Gram schätzen.

3.2.1 n-*Gram*

Ein n-Gram ist vergleichbar mit dem HOC. Für $n = 1$ ist es gleich einem HOC. Bei einem n-Gram enthält jeder Eintrag des korrespondierenden Feature-Vektors die Information, ob ein fest definierter String der Länge n als Teilstring in einem Wort enthalten ist. So sagt ein 1-Gram, oder auch Unigram, aus, ob ein einzelner Buchstabe,

wie *t*, ein 2-Gram, oder auch Bigram, ob ein String der Länge 2, wie *ti* und ein 3-Gram, oder auch Trigram, ob ein String der Länge 3, wie *tio* im Wort enthalten ist. Da, unter Annahme einer vollständigen Abbildung auf alle durch ein Alphabet generierbaren Teilwörter, für ein Alphabet der länge k eine n-Gram Encodierung die Länge k^n besäße, wird für $n > 1$ nach konkreten, vordefinierten, Strings gesucht. Ferner wird vergleichbar zum PHOC eine pyramidiale Struktur von Teilwörtern der Transkription encodiert [PW16].

3.2.2 *Architektur*

Der CNN-Teil von CNN-n-Gram besteht aus 12 konvolutionären Schichten. Eingabebilder haben die feste Skalierung einer Breite von 100, sowie einer Höhe von 32 Pixeln. Die Konvolution einer jeden Schicht wird durch einen 3×3 Kernel mit einer Schrittweite von 1 und der maxout Aktivierungsfunktion [GWFM+13] umgesetzt. Ferner wird nach der dritten, fünften und siebten Schicht ein 2×2 max-pooling mit einer Schrittweite von 2 durchgeführt [PW16].

Im Gegensatz zum PHOCNet wird die Ausgabe des CNN an 19 parallele MLPs, die je ein bestimmtes n-Gram einer spezifischen Ebene der pyramidialen Struktur schätzen, weitergeleitet. Das bedeutet insbesondere für die Ausgabe, dass eine Vielzahl von Attributvektoren ausgegeben wird. Die Attributevektoren umfassen 5 Ebenen von Unigrammen, vergleichbar mit einem PHOC der Tiefe 5, resultierend in 14 Attributvektoren, sowie je 2 Bi- und Trigramme, einer zweiten Ebene. Insgesamt werden also 19 spezifische n-Gramme geschätzt. Die MLPs erhalten dabei die Ausgabe der letzten konvolutionären Schicht und besitzen Schichten der Dimensionalitäten 128, 2048 und der Länge der respektiven n-Gram Encodierung. Dabei verwendet die Ausgabeschicht die Sigmoid-Aktivierungsfunktion und alle vorhergehenden Schichten maxout, siehe 3.2.1. Hierbei ist jedoch zu beachten, dass, wenngleich jedes MLP lokal operiert, durch die 19 parallelen MLPs der Klassifikationsanteil des Modells von CNN-n-Gram vergleichsweise viele Parameter und insbesondere viele Neuronen, wie auch in 3.2.3 thematisiert wird, besitzt. Da die MLPs nicht untereinander verbunden sind, bzw. keine vollständig verbundene Schicht verwendet wird, werden im Folgenden die Schichten ein jedes MLP als lokale Schichten benannt.

Des weiteren führen A. Poznanski und L. Wolf den Begriff der *Neural Codes* ein. Im Fall von CNN-n-Gram beschreiben die Neural Codes dabei die Ausgaben der zweiten lokalen Schicht eines jeden MLPs konkateniert. Diese Ausgaben besitzen mit je 2048 Elementen eine höhere Dimension, als die respektiven n-Gram Encodierungen. Es wird sich erhofft, dass aus dieser größeren Repräsentation, durch die das MLP

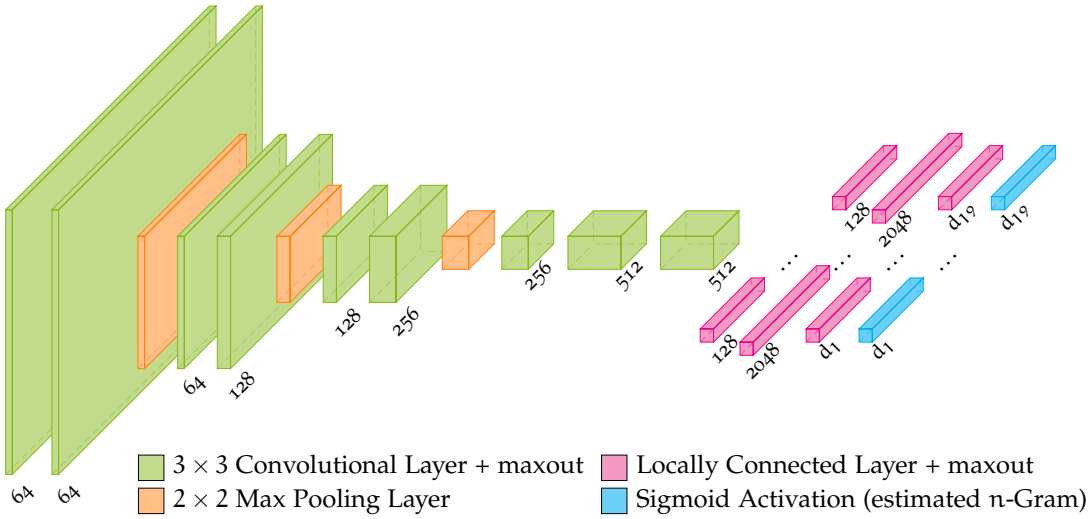

Abbildung 3.2.1: CNN-n-Gram visualisiert. Eingabebilder haben die feste Skalierung einer Breite von 100 und einer Höhe von 32 Pixeln. Nach der letzten konvolutionären Schicht werden die Ausgaben aller Feature-Maps auf 19 MLPs mit 3 Schichten abgebildet, die die n-Grams Kodierungen der respektiven Dimensionen $d_1, \ldots, d_{19}$ schätzen.

das n-Gram schätzt, mehr Informationen gewonnen werden können, als aus dem eigentlichen n-Gram.

3.2.3 *Einzelworterkennung mittels* CNN-n-*Gram*

Für die Einzelworterkennung wird ein Lexikon basierter Ansatz gewählt. Dabei werden die 19 parallel bestimmten Neural Codes als Repräsentation des Wortabbildes gewählt. Diese Repräsentation besitzt $19 \cdot 2048 = 38,912$ Attribute, die durch eine Dimensionsreduktion auf 1200 reduziert werden. Zudem werden binäre Repräsentationen der respektiven n-Grams für alle Lexikon-Wörter berechnet [PW16].

A. Poznanski und L. Wolf wählen den Ansatz einer nächster Nachbar Suche von den reduzierten Repräsentationen der Neural Codes im Lexikon. Hier ergibt sich das Problem, der unterschiedlichen Dimensionalitäten der Repräsentationen und daraus resultierend, dass Distanzmetriken zwischen den Repräsentationen nicht anwendbar sind. Um dennoch mittels einer Distanzmetrik einen nächsten Nachbarn zu suchen wird CCA verwendet, welche die Repräsentation von Neural Codes und n-Grams in einen gemeinsamen Unterraum abbildet. In dem Unterraum wird dann für die Neural

Codes je ein nächster Nachbar aus dem Lexikon mittels der Kosinus-Ähnlichkeit bestimmt.

Dieser Ansatz gleicht dem, des unter dem PHOCNet angewendeten Verfahrens für die Rückgabeliste. Unterschied in diesem Modell ist, dass einzig der erste Eintrag der Rückgabeliste von Bedeutung ist. Ferner besteht die Rückgabeliste aus den Attributvektoren des Lexikons, anstelle von Attributvektoren für Wortabbildungen im Dokument. Das Modell von CNN-n-Gram kann, indem man die für die nächster Nachbar Suche bestimmten Distanzen sortiert auch für das Word-Spotting verwendet werden. Dementsprechend kann auch das PHOCNet für das Problem der Einzelworterkennung angewendet werden.

3.3 PROBABILISTISCHES RÜCKGABEMODELL

Angliedernd an die PHOCNet-Architektur von S. Sudholt und G. A. Fink [SF18] wurde ein probabilistischer Ansatz von E. Rusakov et al. in [RRMF18] für eine Ordnung der Rückgabeliste im Word-Spotting auf Basis von PHOC-Encodierungen vorgestellt. Ziel ist es, *Probabilistic Retrieval Model*-Scores, kurz PRM-Scores, für Wortabbilder hinsichtlich eines Query-Strings (QbS) oder eines geschätzten PHOC für ein Wortabbild (QbE), zu berechnen. Der PRM-Score beschreibt, mit welcher Wahrscheinlichkeit ein Wortabbild das Query-Wort enthält. Um das PRM zu modellieren wurde sich dabei an *Direct Attribute Prediction* [LNH14] orientiert.

3.3.1 *Direct Attribute Prediction*

Damit ein MLP auch Klassen, die nicht im Training vorkamen schätzen kann wurde 2009 in [LNH09], [FEHF09] das Prinzip der Attribut basierten Klassifikation eingeführt. Ein MLP schätzt Attribute einer Klasse, anstelle direkt zu Klassifizieren. Dies führt dazu, dass auch für Klassen, die nicht im Training vorkamen, auf deren Attribute abgebildet werden kann. Bei einem Vergleich dieses Ansatzes mit dem PHOCNet, ist die Repräsentation ein PHOC und das MLP entspricht dem MLP Teil des PHOCNet. Die Klassifikation von Klassen, die nicht im Training vorkamen wird dabei auch als *zero-shot learning* [LNH09], [LNH14] bezeichnet. *Direct Attribute Prediction*, kurz DAP, wurde angelehnt an die Attribut basierte Klassifikation von Lambert et al. [LNH09]. Ziel hierbei ist es über Attribute, als Zwischeninstanz, auf eine Klassifikation zu schließen, anstelle einer direkten Klassifikation, siehe Abbildung 3.3.1.

DAP beschreibt die A-posteriori-Wahrscheinlichkeit $p(z|x)$, also der Wahrscheinlichkeit der Zugehörigkeit zu einer Klasse z, gegeben einer Eingabe x. Um $p(z|x)$ zu

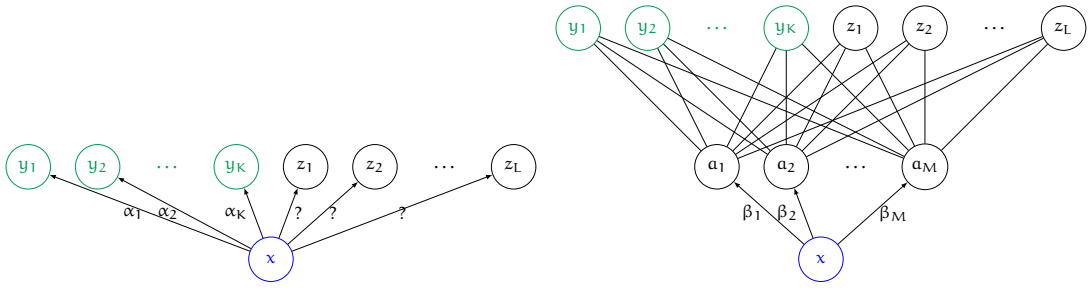

Abbildung 3.3.1: Direkte Klassifikation (links): Ein Klassifikator lernt in Abhängigkeit einer Eingabe x eine Menge von Parametern $\{\alpha_1, \ldots, \alpha_K\}$ für die aus dem Training bekannten Klassen $Y = y_1, \ldots, y_K$. Problem hierbei ist, dass der Klassifikator nicht auf die Klassen $Z = z_1, \ldots, z_L$, die nicht im Training vorkamen, abbilden kann.
DAP (rechts): Anstelle von Klassen werden vom Klassifikator Attribute $a = a_1, \ldots, a_M$ geschätzt. Sowohl Y, als auch Z können für ihre Elemente Repräsentationen, durch eine Kodierung der Attribute aus a, erzeugen. Auf Basis der Kodierungen ist es möglich eine aus dem Training unbekannte Klasse $z \in Z$ zu schätzen. Abbildungen 2 (a), (b) aus [LNH14].

modellieren wird der Attributvektor $a = a_1, \ldots, a_M$ betrachtet. Die Attribute aus a werden als eine Art Zwischeninstanz verwendet, um auf $p(z|x)$ über $p(z|a)$ und $p(a|x)$ zu schließen. In Abbildung 3.3.1 ist visualisiert, wie der Attributvekotor eine Zwischenschicht der Klassifikation bildet, die eine Eingabe mit allen Klassen $Y \cup Z$ und insbesondere den aus den Training unbekannten Klassen Z des Datensatzes verbindet und ferner eine Abbildung auf Klassen aus Z ermöglicht. Nach dem in Abbildung 3.3.1 visualisiertem Schichtenmodell benötigen wir also die A-posteriori-Wahrscheinlichkeiten für $p(z|a)$ und $p(a|x)$ um durch

$$p(z|x) = \sum_{a \in \{0,1\}^M} p(z|a)p(a|x) \tag{3.3.1}$$

ein Modell für $p(z|x)$ zu bestimmen.

Betrachten wir zunächst $p(a|x)$. Gegeben sei ein Klassifikator, der gegeben einer Eingabe x die Wahrscheinlichkeit der Vorkommnis eines jeden Attributes $a_m, m \in \{1, \ldots, M\}$ in x schätzt. Ferner habe jede Klasse y das binäre Label a_m^y, dass das

Vorkommnis des respektiven Attributes impliziert. Im wesentlichen realisiert also der Klassifikator das Schätzen von $p(a_m|x)$, woraus mit

$$p(a|x) = \prod_{m=1}^{M} p(a_m|x) \tag{3.3.2}$$

ein Modell für alle geschätzten Attribute erstellt werden kann.

Ferner sei der Attributvektor a^z einer Klasse z deterministisch berechenbar. Daraus folgt direkt $p(a|z) = [\![a = a^z]\!]$, wobei die Iversons Klammernotation einen booleschen Wert auf 1 für wahr und 0 für falsch abbildet [Knu92]. Hinsichtlich $p(z|a)$ folgt aus dem Satz von Bayes wiederum:

$$p(z|a) = \frac{p(z)}{p(a^z)}[\![a = a^z]\!] \tag{3.3.3}$$

Nun, da Therme für $p(z|a)$ und $p(a|x)$ bestimmt wurden erhalten wir durch einsetzen von 3.3.2, 3.3.3 in 3.3.1

$$p(z|x) = \frac{p(z)}{p(a^z)} \prod_{m=1}^{M} p(a_m^z|x), \tag{3.3.4}$$

unter Berücksichtigung, dass $[\![a = a^z]\!]$ nur für den Attributvektor a^z ungleich 0, bzw. gleich 1 ist und damit restliche Summanden wegfallen. Durch Annahme identischer A-priori-Wahrscheinlichkeit der Klassen und Unabhängigkeit für alle $a_m, m \in \{1, \ldots, M\}$, kann ferner $p(z)$ vernachlässigt und für $p(a)$ eine faktorielle Verteilung $p(a) = \prod_{m=1}^{M} p(a_m)$ angenommen werden. Daraus resultiert dann die vereinfachte Formel

$$p(z|x) = \prod_{m=1}^{M} \frac{p(a_m^z|x)}{p(a_m)} \tag{3.3.5}$$

für $p(z|x)$.

Um nun die respektiv zu einer Eingabe x, beste Klasse aus $z_1, \ldots, z_L$ zu bestimmen, wird die Entscheidungsfunktion

$$f_{pred}(x) = \underset{l \in \{1,\ldots,L\}}{\operatorname{argmax}}\, p(z|x) = \underset{l \in \{1,\ldots,L\}}{\operatorname{argmax}} \prod_{m=1}^{M} \frac{p(a_m^{z_l}|x)}{p(a_m^{z_l})} \tag{3.3.6}$$

für einen auf DAP basierten Klassifizierungsansatz verwendet [LNH14].

3.3.2 *Probabilistic Retrival Model-Scores*

Vergleichbar zum Vorgehen bei DAP suchen Rusakov et al. nach der Wahrscheinlichkeit, dass für eine Query q das Eingabebild x relevant ist. Analog zur Methodik von DAP, lässt sich dann das folgende Modell für die A-posteriori-Wahrscheinlichkeit herleiten [RRMF18]:

$$p(q|x) = \sum_{a \in \{0,1\}^M} p(q|a)p(a|x) \tag{3.3.7}$$

Durch das Anwenden des Satz von Bayes auf $p(q|a)$ erhalten wir $p(q|a) = \frac{p(q)}{p(a)}p(a|q)$. Da $p(a|q)$ nur für den zum PHOC der Query korrespondierenden Attributvektor a^q ungleich 0, bzw. gleich 1 ist erhalten wir durch einsetzen in 3.3.7:

$$p(q|x) = \frac{p(q)}{p(a^q)}p(a^q|x) \tag{3.3.8}$$

Rusakov et al. vernachlässigen die A-priori-Wahrscheinlichkeiten $p(q), p(a^q)$ gänzlich. Grund dafür ist, dass für die simplifizierte Formel 3.3.5 des DAP nach Lambert et al. [LNH14] die A-priori-Wahrscheinlichkeit auch weitestgehend vernachlässigt wurden und dies zu guten Ergebnissen führte. Ferner ist es nicht ersichtlich, wie eine A-priori-Wahrscheinlichkeit über eine unbekannte Menge von Queries direkt abgeleitet werden kann. Die A-posteriori-Wahrscheinlichkeit für $p(q|x)$ wird also zu

$$p(q|x) = p(a^q|x) \tag{3.3.9}$$

vereinfacht und das Modell reduziert sich auf das Bestimmen von $p(a|x)$.

Für die Bestimmung von $p(a|x)$ wird darauf aufgebaut, dass das PRM für das PHOCNet, dessen Ausgabeschicht insbesondere eine Sigmoid-Aktivierungsfunktion realisiert, entwickelt wurde. Dadurch kann, um $p(a|x)$ zu bestimmen, ausgenutzt werden, dass durch die Sigmoid-Aktivierungsfunktion der Ausgabeschicht des PHOCNet jedes Element $\hat{a}_i$, des ausgegebenen Attributvektor $\hat{A}$, als die geschätzte Wahrscheinlichkeit, dass das i-te Attribut im Eingabebild x vorkommt, interpretiert werden kann:

$$p(a|x) = \prod_{i=1}^{D} p(A_i = a_i|x) \tag{3.3.10}$$

$$= \prod_{i=1}^{D} \hat{a}_i^{a_i} \cdot (1 - \hat{a}_i)^{(1-a_i)}, \text{ mit } \hat{a}_i = p(A_i = 1|x) \tag{3.3.11}$$

Hierbei ist also a das PHOC des Query-Strings und x ein Wortabbild. A_i bezeichnet das i-te Attribut des PHOC der Transkription von x. Wir interpretieren den Attributvektor dabei als die Wahrscheinlichkeiten, dass A_i im PHOC der Transkription enthalten, bzw. dessen Wert gleich 1, ist. In anderen Worten, der Attributvektor entspricht $\hat{A} = (p(A_1 = 1|x), \ldots, p(A_N = 1|x))^T$, mit der Länge N des respektiven PHOC. Wie in 3.3.11 definiert multiplizieren wir also die Wahrscheinlichkeiten, dass die respektiven Attribute gleich sind, um die Gesamtwahrscheinlichkeit zu erhalten, dass x für a relevant ist.

Hinsichtlich einer Implementierung des PRM, ergibt sich für PHOC mit hoher Dimension das Problem der numerischen Stabilität, da die aus der Produktformel resultierenden Werte zu klein werden können, als dass sie als Gleitkommazahl hinreichend genau darstellbar wären. Für die Realisierung numerischer Stabilität wurde von Rusakov et al. deshalb mit dem Logarithmus eine streng monotone Funktion auf den PRM-Score angewandt, die die Ordnung der Rückgabeliste jedoch, aufgrund der strengen Monotonie, nicht beeinflusst [RRMF18]:

$$\log(p(q|x)) = \log(\prod_{i=1}^{D} \hat{a}_i^{a_i} \cdot (1 - \hat{a}_i)^{(1-a_i)}) = \sum_{i=1}^{D} a_i \log(\hat{a}_i) + (1 - a_i) \log(1 - \hat{a}_i) \tag{3.3.12}$$

Da der Logarithmus für 0 eine Definitionslücke aufweist und sowohl Wahrscheinlichkeiten $\hat{a}_i$, als auch Gegenwahrscheinlichkeiten $(1 - \hat{a}_i)$ stets größer gleich 0 sind, bietet es sich ferner an eine infinitesimale, positive Konstante ϵ auf alle Wahrscheinlichkeiten und Gegenwahrscheinlichkeiten in 3.3.12 zu addieren.

Nehmen wir nun an Wortlassen $c_1, \ldots, c_H$ seien gegeben. So sei $A^{c_h} = (a_1^{c_h}, \ldots, a_1^{c_h})^T$ der Attributvektor zum PHOC der h-ten Wortklasse und $\hat{A} = (\hat{a}_1, \ldots, \hat{a}_1)^T$ der geschätzte Attributvektor, gegeben einer Eingabe x. Dann folgt aus den PRM-Scores die Entscheidungsfunktion des Probabilistischen Modells, analog zur Entscheidungsfunktion 3.3.6 der DAP:

$$f_{PRM}(x) = \underset{h \in \{1,\ldots,H\}}{\operatorname{argmax}} \; p(c_h|x) = \underset{h \in \{1,\ldots,H\}}{\operatorname{argmax}} \sum_{i=1}^{D} a_i^{c_h} \log(\hat{a}_i) + (1 - a_i^{c_h}) \log(1 - \hat{a}_i) \tag{3.3.13}$$

Gleichung 3.3.12 kann auch als zwei Summen:

$$\log(p(q|x)) = \sum_{i=1}^{D} a_i \log(\hat{a}_i) + \sum_{i=1}^{D} (1 - a_i) \log(1 - \hat{a}_i) \tag{3.3.14}$$

dargestellt und damit $\log(p(q|x))$ als die Summe der Skalarprodukte $A^T \log(\hat{A})$ und $(1-A)^T \log(1-\hat{A})$ betrachtet werden. Das Modell des PHOCNet verwendet nativ die Kosinus-Ähnlichkeit des eingeschlossenen Winkels α zweier Repräsentationen $A, \hat{A}$

$$\cos(\alpha) = \frac{\sum_{i=1}^{D} a_i \hat{a}_i}{\sqrt{\sum_{i=1}^{D} a_i^2} \sqrt{\sum_{i=1}^{D} \hat{a}_i^2}} = \frac{A^T \hat{A}}{\|A\|_2 \|\hat{A}\|_2}, \tag{3.3.15}$$

durch welche auch das Skalarprodukt $A^T \hat{A} = \|A\|_2 \|\hat{A}\|_2 \cos(\alpha)$ dargestellt werden kann. Die PRM-Scores berücksichtigen indirekt also neben den aus den Wahrscheinlichkeiten $\hat{A}$, auch die aus den Gegenwahrscheinlichkeiten $1 - \hat{A}$ resultierenden Skalarprodukte.

4

METHODIK

4.1 EINZELWORTERKENNUNG DURCH PHOCWRec

Für die Einzelworterkennung wird ein ähnlicher Ansatz, wie in [PW16] verwendet. Eine Menge von Eingaben X soll auf eine Menge von Klassen C abgebildet werden. Hierbei bildet sich X aus Wortabbildern und C aus den Wörtern eines Lexikons, bzw. Wortklassen des Datensatzes. Im Lexikon sind alle Wörter des Datensatzes, insbesondere aus Trainings- und Testmenge, enthalten. Um die Abbildung $f_{wrec} : X \rightarrow C$ zu realisieren, führen wir mit der Menge von möglichen Attributvektoren A eine Zwischenschicht im Modell ein, siehe Abbildung 4.1.1.

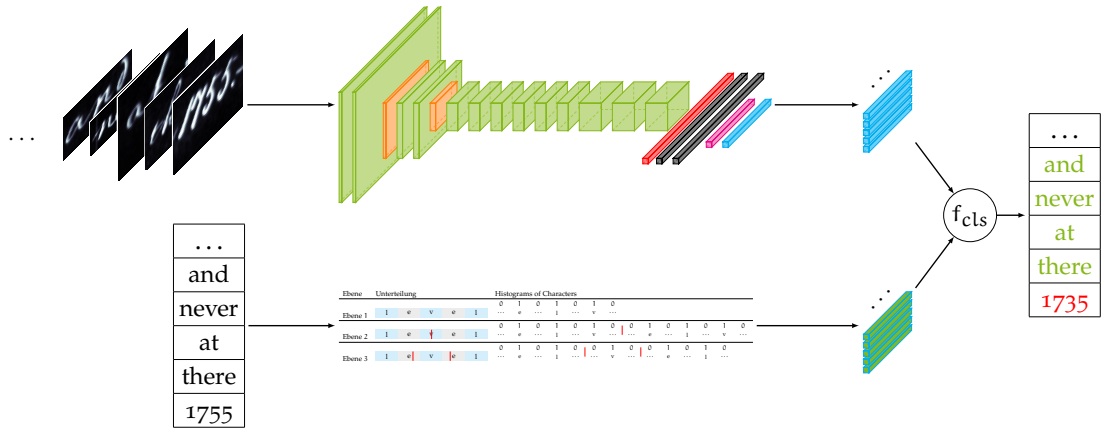

Abbildung 4.1.1: Der Ablauf der Einzelworterkennung, wie in dieser Arbeit vorgestellt. Mittels einer Instanz des PHOCNet werden für Wortabbilder je ein PHOC geschätzt, während für ein Lexikon respektive PHOC deterministisch bestimmt werden. Unter Eingabe der geschätzten PHOC und der PHOC des Lexikons realisiert ein Modell mit der Entscheidungsfunktion f_{cls} die Klassifikation der geschätzten PHOC zu einer Wortklasse.

Es werden Wortabbilder X und Wortklassen C auf Attributvektoren aus A abgebildet. Im folgenden werden Attributvektoren der Eingaben mit A_X und Attributvektoren der Wortklassen mit A_C bezeichnen. Das Schätzen einer Wortklasse erfolgt dann über ein mathematisches Modell, mit der Entscheidungsfunktion $f_{pred} : A_X \rightarrow C$,

unter Kenntnis der Attributvektoren A_C für die Wortklassen. Der Attributvektor sei ein PHOC mit 3 Ebenen. Um Wortabbilder in deren Attributvektoren A_X zu überführen verwenden wir eine trainierte Instanz des PHOCNet. Die Abbildung von Wortklassen auf deren Attributevektoren A_C folgt direkt aus der Definition des PHOC. Das beschriebene Modell wird im Folgenden als *PHOC Word Recognition*, kurz PHOC*W*Rec bezeichnet.

4.2 LEXIKON

In PHOC*W*Rec wird, für einen geschätzten Attributvektor, mittels der Entscheidungsfunktion eines Klassifikators, die Wortklasse geschätzt. Damit Wörter des Testdatensatzes durch PHOC*W*Rec erkannt werden können setzten wir daher die Menge aller Transkriptionen der Trainingsdaten C_{train} und Testdaten C_{test} als bekannt voraus. Das für PHOC*W*Rec verwendete Lexikon sei dementsprechend als die Vereinigung von Trainings- und Testwortklassen $C = C_{train} \cup C_{test}$ definiert. Dies ist notwendig, damit auch auf Wortklassen aus C_{test}, die nicht in C_{train} enthalten sind, abgebildet werden kann. Somit vermeiden wir *out of vocabulary* Wortklassen. Die Berechnung der zu den Klassen respektiven PHOC folgt aus der Definition des PHOC.

4.3 ATTRIBUTERKENNUNG MITTELS PHOCNET

Im Folgenden wird erörtert, wie das PHOCNet in PHOC*W*Rec verwendet wird. Dazu werden Eingabebilder, Pyramidal Pooling, die Attribut Repräsentationen der PHOC, sowie die Initialisierung, das Training und die Regularisierung des PHOCNet diskutiert.

4.3.1 *Eingabebilder*

Hinsichtlich der Eingabebilder sind, über die Architektur des PHOCNet, keine Einschränkungen der Repräsentation und Skalierung gegeben. Im Folgenden werden Repräsentation und Skalierung der Wortabbilder diskutiert.

Repräsentation der Wortabbilder

Es wird angenommen, dass alle Wortabbilder in schwarz-weiß vorliegen. Unter dieser Annahme können alle Wortabbilder variabler Höhe h und Breite w mittels einer Matrix

$I \in [0,1]^{h \times w}$ repräsentiert werden, deren Werte die Helligkeit eines jeweiligen Pixel implizieren. Ferner werden alle Wortabbilder durch

$$f_{bright}(I) = \frac{I}{\max I} \tag{4.3.1}$$

hinsichtlich ihrer Helligkeit insoweit skaliert, als dass ihr maximaler Wert stets gleich 1 ist. Zudem erfolgt eine Invertierung der Wortabbilder $I' = 1 - I$. Die Invertierung ist darin Motiviert, dass bei schwarz-weiß Bildern Buchstaben schwarz und Papier weiß abgelichtet werden. Durch die Invertierung werden die Werte der Buchstaben anstelle derer des Hintergrundes propagiert. Diese Repräsentation der Wortabbilder wurde von S. Sudhold und G. A. Fink in [SF18] vorgestellt.

Skalierung

Durch eine Instanz des *Pyramidal Poolings* ist es dem PHOCNet möglich Bilder beliebiger Skalierung zu prozessieren. Würde ein Verfahren angewandt werden, dass nicht relativ zur Größe der Feature-Maps auf die erste Schicht des MLPs abbildet, müsste jede Eingabe einer festen Skalierung entsprechen. Möglichkeiten dies für Eingabebilder, die nicht dieser Skalierung entsprechen, umzusetzen wären das Ausschneiden eines Bildbereiches und Stauchen, bzw. Strecken des Eingabebildes. Beim Ausschneiden eines Bildbereiches werden inhärent Bildabschnitte und damit Informationen ignoriert. Dieses Verfahren wird deshalb nicht weiter in Erwägung gezogen. Durch das Stauchen und Strecken von Eingabebildern werden keine Bildabschnitte ignoriert, jedoch wird die räumliche Verteilung von Bildmerkmalen, wie etwa Buchstaben, auf eine feste Skalierung normiert.

Da unter Verwendung einer Instanz des Pyramidal Poolings Bilder beliebiger Skalierung verwendet werden können, jedoch z.B. beim TPP nur temporal und mit einer festen Anzahl von Bins gepoolt wird, ergibt sich die Frage in wie weit die Originalskalierung eines Bildes besser geeignet ist, als ein fest skaliertes Bild einer angemessenen Größe. In [RSS+18] wurden bereits Eingabebilder hinsichtlich ihrer Skalierung für das Word-Spotting untersucht. Neben dem schnellerem Training bei Bildern mit einer fixen Skalierung, erzielten Modelle, die fix skalierte Eingaben besaßen bessere Ergebnisse. Aufgrund dessen werden PHOCNet Instanzen, die auf verschiedenen Skalierungen trainiert wurden, in Betracht gezogen.

4.3.2 *Pyramidal Pooling*

Vergleichbar zur in [SF18] für das Problem des Word-Spottings vorgestellten Konfiguration eines PHOCNet, verwenden wir eine Instanz des PHOCNet die ein Pyramidal

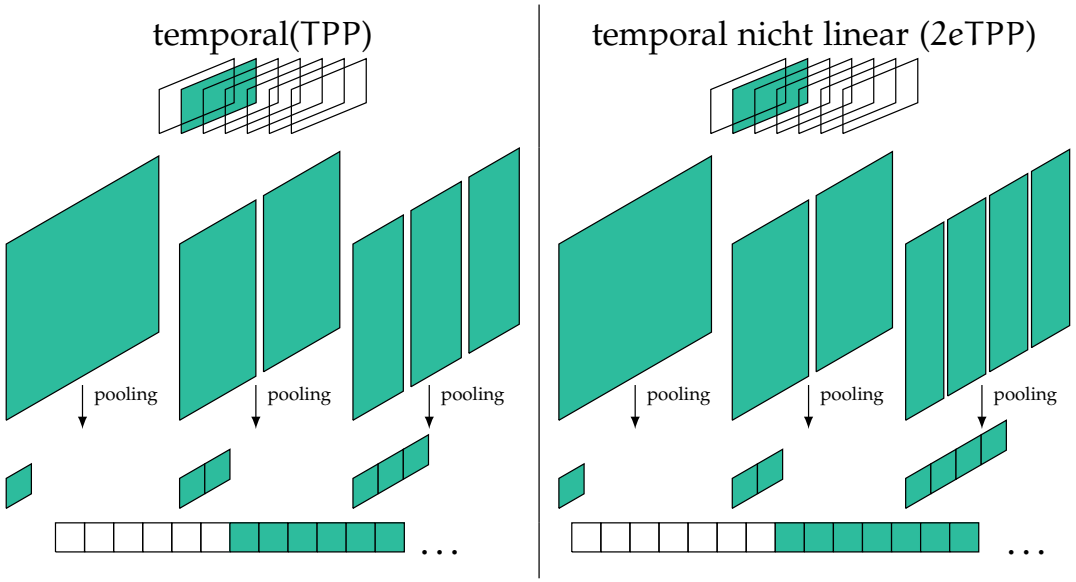

Abbildung 4.3.1: TPP und 2eTPP für eine Feature-Map visualisiert.

Pooling Verfahren verwendet. Hinsichtlich des pyramidialen Poolings nutzten wir ein an das TPP angelehntes Verfahren. Wie auch beim TPP wird temporal gepoolt, jedoch nimmt die Anzahl an Spalten mit 2^t für die t-te Ebene exponentiell zu, wie auch beim SPP, siehe Abbildung 4.3.1. Aufgrund des, zur Basis 2, exponentiellen Wachstums der Spalten je Ebene, referenzieren wir zu dieser Art von TPP im folgenden als *2-exponential Temporal-Pyramidal-Pooling*, kurz 2eTPP.

Es wird eine 2eTPP Schicht mit 5-Ebenen verwendet, deren Ausgabe ein Vektor mit $512 \cdot \sum_{t=0}^{4} 2^t = 15872$ Werten ist. Im Vergleich zu den $512 \cdot \sum_{t=1}^{4} t = 5120$ Werten des TPP mit Tiefe 5 und $512 \cdot \sum_{t=0}^{2} 4^t = 10752$ Werten des SPP mit Tiefe 3 erhalten wir damit eine vergleichsweise große Anzahl von repräsentativen Werten für temporale Ausschnitte der Feature-Maps.

4.3.3 *Attribut Repräsentationen*

Jeder Attributvektor aus A_C ist ein PHOC mit 3 Ebenen. Betrachtet man ein PHOC als Attributvektor, so ist ein wichtiger Aspekt das Alphabet, aus dem sich die einzelnen HOC ergeben. Fehlen im Alphabet ein oder mehrere Buchstaben für eine Transkription, so können diese Buchstaben auch in keinem HOC als Attribut encodiert werden. Dies kann zu falschen Ausgaben der Entscheidungsfunktion f_{cls} führen, da gegebenen falls

das resultierende PHOC dem einer anderen Wortklasse gleicht. Ein weiterer Aspekt ist die Länge des Alphabets. Je höher die Dimensionalität der Attributvektoren, desto schwieriger sind diese, z.B. durch Distanzmetriken, zu vergleichen. Es sollten generell nicht mehr und nicht weniger Buchstaben in ein Alphabet aufgenommen werden als nötig. Eine Möglichkeit wäre dementsprechend auf alle im Datensatz enthaltene Buchstaben abzubilden, bzw. die HOC auf einem Alphabet zu basieren, dass alle relevanten Buchstaben enthält. Dies bewirkt, dass das resultierende PHOCNet auf den Datensatz zugeschnitten ist. Wir betrachten auch Zusammensetzungen aus generischen Buchstabensätzen, siehe Abbildung 4.3.2. Im Rahmen dieser Buchstabensätze sind auch Alphabete ohne Großbuchstaben möglich. Für diesen Fall werden alle Großbuchstaben von Trainings- und Testdaten auf ihre respektiven Kleinbuchstaben abgebildet. Es werden also Ausgaben akzeptiert, bei denen ein Buchstabe hätte groß geschrieben werden müssen. Damit greifen wir streng genommen in die Komplexität des Problems der Einzelworterkennung ein. Jedoch implizieren Wörter, wie *Then, then* die selbe Wortklasse. Einziger Unterschied ist, dass eines der Wörter zu Satzbeginn geschrieben, oder in einer Überschrift verwendet wird. Da beide Wörter die selbe Bedeutung, bzw. Semantik haben erlauben wir diese Reduzierung der Wortklassen.

Buchstabensatz	Alphabet
lower-case (L)	abcdefghijklmnopqrstuvwxyz
upper-case (U)	ABCDEFGHIJKLMNOPQRSTUVWXYZ
digits (D)	0123456789
punctuation (P)	!"#$%& ’()*+,-./:;<=>?@[\]^_ ‘|~

Abbildung 4.3.2: Alphabete der allgemeinen Buchstabensätze.

4.3.4 *Initialisierung des Modells*

Wie im Teil 2.3 zur Optimierung neuronaler Netze diskutiert, finden SGD oder auch andere Optimierungsverfahren ein lokales Minimum der Fehlerfunktion für Parameter Θ eines Modells. Die Initialisierung eines neuronalen Netzes beeinflusst dabei inhärent welches lokale Minimum, wie schnell, gefunden wird und ferner das optimierte Modell. In [SF18] diskutieren S. Sudhold und G. A. Fink die Bedeutung der Initialisierung des PHOCNet für dessen Training. Dabei verweisen sie auf ein in [HZRS15a] vorgestelltes Verfahren, bei dem alle Biase mit 0 und alle Gewichte aus einer Normalverteilung

mit Standardvarianz $\frac{2}{n}$, mit der Anzahl von Eingehenden Kanten eines Filters n, initialisiert werden.

Wir adaptieren diesen Ansatz der Initialisierung des PHOCNet genau dann, wenn kein vortrainiertes PHOCNet verwendet wird. Vortrainierte, oder auch *pretrained*, neuronale Netze wurden bereits auf einem Datenbestand angelernt. Man erhofft sich von diesen Modellen, dass sie schneller optimiert werden können und in Folge dessen bei gleicher Anzahl von Iterationen bessere Resultate erzielen, als Modelle, die randomisiert initialisiert wurden.

4.3.5 *Training*

Für die Experimente werden Instanzen des PHOCNet mit einer 2*e*TPP Schicht der Tiefe 5 und Ausgabevektoren die der Dimensionalität eines PHOC mit 3 Ebenen und einem beliebigem Alphabet entsprechen trainiert. Hinsichtlich der Optimierungsschritte wird ein Mini-Batch mit 10 Wortabbildern verwendet. Für vergleichsweise kleine Datensätze umfasst das Training 100000 Iterationen, für größere Datensätze 200000. Die Anzahl der Iterationen wird in 5.1 für jeden Datensatz spezifiziert.

Optimierungsverfahren

Hinsichtlich der Optimierung wird nicht SGD, sondern Adptive Momentum (*ADAM*) [KB14] als Optimierungsverfahren verwendet. ADAM, basiert auf dem verfahren von Adaptive Gradient (*AdaGrad*) [DHS11]. AdaGrad, passt die Learning-Rate online, in Abhängigkeit der vergangenen Gradienten, an. Sei $g_t = \nabla C_t(\Theta)$ der Vektor zur partiellen Ableitung einer Fehlerfunktion C nach dem t-ten Mini-Batch. Dann umfasst $G_t = \sum_{i=1}^{t} g_i g_i^T$, die Summe der dyadischen Produkte aller bisher erfassten Gradientenvektoren. Insbesondere sind auf der Diagonalen von G die Summen der quadrierten Gradienten enthalten. Folglich erhalten wir durch:

$$\Theta_{t+1,i} = \Theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii}} + \varepsilon} g_{t,i} \tag{4.3.2}$$

ein Verfahren, dass für jeden Parameter Θ_i die Learning-Rate respektiv angleicht. Hierbei ist ε eine Konstate größer, aber nahe 0. Dabei wird ε addiert um einer Nulldivision vorzubeugen. Für Parameter, mit frequent starken Gradienten erfolgen dadurch kleinere und für Paramater mit infrequent starken Gradienten größere Optimierungsschritte.

Obwohl AdaGrad eine Möglichkeit bietet die Learning-Rate für jeden Parameter Anzupassen, birgt es den Nachteil, dass insbesondere für eine hohe Anzahl von Iterationen die Summen der quadrierten Gradienten zu groß und damit die

respektiven angeglichenen Learning-Raten zu klein werden. ADAM verfolgt auch den Ansatz, die Learning-Rate für jeden Parameter respektiv anzupassen. Jedoch betrachten wir für ADAM mit dem geschätztem durchschnittlichen Gradientenvektor $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$ und der geschätzten Varianz der Gradientenvektoren $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$ das erste und zweite stochastische Moment, anstelle der quadrierten Gradienten, wobei $\beta_1, \beta_2 \in [0,1)$ die Parameter des exponentiellen Verfalls der geschätzten Momente sind. Die geschätzten stochastischen Momente werden im Optimierungsschritt wie folgt berücksichtigt:

$$\hat{m}_t = \frac{m_t}{(1-\beta_1^t)} \tag{4.3.3}$$

$$\hat{v}_t = \frac{v_t}{(1-\beta_2^t)} \tag{4.3.4}$$

$$\Theta_{t+1} = \Theta_t - \frac{\eta \hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} \tag{4.3.5}$$

Wir verwenden ADAM mit den Hyperparameter $\beta_1 = 0,9; \beta_2 = 0,999$ für die Optimierung des PHOCNet. Als Fehlerfunktion wird die Kosinus-Fehlerfunktion 2.3.3 verwendet. Des weiteren wird die Learning-Rate initial auf $\eta = 0,0001$ gesetzt.

Dieses Trainingsverfahren wird auch in [SF18] vorgeschlagen. Ferner dividieren wir die Learning-Rate nach 60000 Iteration durch 10, wie in [RSS+18] vorgeschlagen, was zu einer schnelleren Optimierung führt.

Regularisierung

Um Overfitting zu vermeiden und die Invarianz des neuronalen Netzes gegenüber kleiner Varianz der Eingaben zu erhöhen werden mit einer Augmentierung der Trainingsdaten, *weight-decay* und *Dropout* Regularisierungsverfahren angewandt.

In [SF18] wurde für das Training des PHOCNet ein Augmentierungsverfahren vorgestellt. Dabei werden zufällige affine Transformationen an Wortabbildern Vorgenommen um mehr Wortabbilder für die im Trainingssatz vorkommenden Wortklassen zu generieren. Die Affine Transformation leitet sich aus 3 Bildpunkten, siehe Abbildung 4.3.3, her und die Verteilung der Wortklassen kann als gleichverteilt, oder ungleichverteilt angenommen werden. Bei einer ungleichen Verteilung wird die relative Häufigkeit einer jeden Wortklasse in der Augmentierung bewahrt, während bei einer Gleichverteilung alle Wortklassen die selbe relative Häufigkeit besitzen. Über die Augmentierung erhalten wir mehr Varianz in den Wortabbildern, das optimierte Modell sollte gegenüber Rotation oder schlecht segmentierten Wortabbildern invarianter sein. Ferner kann über die Augmentierung eine Verteilung von Wortklassen im

Trainingsdaten festgelegt werden. Durch die Augmentierung wird des weiteren die Invarianz des neuronalen Netzes gegenüber Rotation und auch sonstiger räumlicher Varianz der Position von Wortabbildern im segmentierten Bildbereich erhöht.

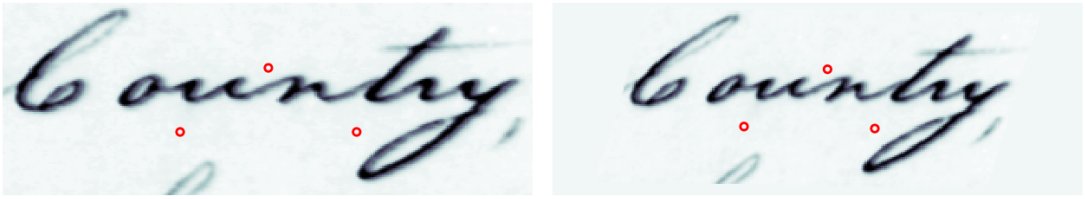

Abbildung 4.3.3: Drei Punkte mit fixen relativen Abständen (links) werden entnommen und je mit einer Stichprobe aus einem gleich verteilten Intervall $[0,8; 1,1]$ multipliziert. Auf Basis der aus den Produkten resultierenden Punkte (rechts) wird eine affine Transformation an dem Eingabebild vorgenommen [SF18]. Bild entnommen aus der George Washington Datenbank [RM07].

Bei dem weight-decay wird versucht Overfitting zu vermeiden. Gewichte, die sonst keine Änderungen im Optimierungsschrit erhalten konvergieren durch weight-decay exponentiell gegen Null. Dies wird im Optimierungsschritt

$$w_i = w_i - \eta \frac{\delta C}{\delta w_i} - \eta \lambda w_i \tag{4.3.6}$$

durch den Faktor λ realisiert. Für das Training aller Modelle wird ein weight-decay von $\lambda = 0,0005$ verwendet, wie auch in [SF18] vorgeschlagen.

Damit das neuronale Netz ferner keine Pfade für die Eingabebilder bevorzugt, wird Dropout [SHK+14] verwendet. Dropout wird für alle Schichten nach der 2eTPP Schicht bis ausschließlich zur vorletzten Schicht verwendet. Bei Dropout wird ein jedes Neuron eine Schicht mit einer Wahrscheinlichkeit p deaktiviert, bzw. dessen Ausgabe durch 0 ersetzt. Dadurch wird das MLP dazu gezwungen die Attribut-Klassifikation über verschiedene Pfade zu lernen. Im Wesentlich wird durch Dropout realisiert, dass das MLP nicht als ein einzelner Attribut-Klassifikator verstanden werden muss, sonder eher als eine Menge von MLPs die ihre Gewichte teilen [SF18]. Dies reduziert die Wahrscheinlichkeit, dass das neuronale Netz auf bestimmte Aktivierungen angewiesen ist und verstärkt die Invarianz gegenüber verrauschten Daten.

4.4 KLASSIFIKATOREN

Für die Realisierung von f_{cls} betrachten wir verschiedene Modelle. Im folgenden setzten wir voraus, dass für jede Wortklasse $c \in C$ eine Zuordnung zu genau einem

Attributvektor $a_C \in A_C$ und umgekehrt über eine Indizierung existiert. Da es nicht garantiert ist, dass das PHOC einer jeden Wortklasse einzigartig ist, in anderen Worten die Abbildung auf ein PHOC nicht bijektiv ist, bedeutet dies, dass gleiche Attributvektoren $a_j, \ldots, a_k \in A_C : a_j = \ldots = a_k$ nach dieser Zuordnung nicht zwingend der gleichen Wortklasse angehören. Die Klassifikatoren beruhen darauf, gegeben des geschätzten Attributvektors und den Attributvektoren des Lexikons deterministisch Werte, die die Ähnlichkeit der verglichenen Vektoren beschreiben, zu berrechnen. Für gleiche Attributvektoren mit verschiedenen Wortklassen kann dies zu Klassifizierungsfehlern führen. Es kann keine Aussage getroffen werden welche Klasse aus $c_j, \ldots, c_k$ geschätzt wird. Daher ist es wichtig ein hinreichend tiefes PHOC als Repräsentation der Attributvektoren zu wählen, um solche Klassifizierungsfehler zu vermeiden. Wie in 4.3.3 erwähnt verwenden wir ein PHOC mit 3 Ebenen, wie auch in [SF18], [RSS+18] vorgeschlagen. Dies wird im Rahmen dieser Arbeit als hinreichend tief erachtet und die Auswirkung der Tiefe des PHOC auf die Klassifikatoren wird nicht weiter untersucht.

Eine Möglichkeit auf Basis der Attributvektoren zu klassifizieren ist mittels einer nächster Nachbar Suche:

$$\underset{a_C \in A_C}{\operatorname{argmin}} f_{dist}(a_C, a_x) \tag{4.4.1}$$

Hierbei vergleichen wir einen geschätzten Attributvektor $a_X \in A_X$ mit allen Attributvektoren der Wortklassen A_C durch eine geeignete Distanzmetrik f_{dist} und erhalten den Index des ähnlichsten Attributvektors aus A_C, über den wir die Wortklasse herleiten.

Eine weitere Möglichkeit bietet die in 3.3.13 vorgestellte Entscheidungsfunktion von Rusakov et al. [RRMF18]. Das auf DAP aufbauende verfahren der PRM-Scores ist direkt auf dieses Modell der Einzelworterkennung anwendbar, da durch A_X die geschätzten Attributvektoren gegeben sind und A_C wie die in 3.3 angesprochenen Attributvektoren der Klassen behandelt werden können.

Die letzte Möglichkeit der attributbasierten Klassifikation, die untersucht wird, ist die durch CCA. Wie in [PW16] vorgestellt kann es sich als Sinnvoll erweisen mittels CCA für die Ausgabe einer Schicht eines neuronalen Netzes und Kodierungen eines Lexikons einen gemeinsamen Unterraum zu finden. Da, wie auch in 3.2.3 angesprochen, die Ansätze von PHOCNet und CNN-n-Gram Ähnlichkeiten aufweisen werden wir untersuchen, in wie fern die Integration von CCA in das Modell PHOCWRec Verbesserungen aufweist. Der Ansatz von [PW16] basiert auf Neural Codes, genauer den Ausgaben der zweiten Schicht eines jeden lokalen MLPs. Betrachten wir Abbildung 4.4.1, so sehen wir, dass im Modell des PHOCNet das MLP im Wesentlichen eine Dimensionsreduktion durchführt, um dann mittels der Sigmoid-Aktivierungsfunktion

ein PHOC zu schätzen. Dabei dient die vorletzte Schicht einer Projektion von zuvor, durch ReLU, stets positiven Daten auf einen Wertebereich, der mit positiven und negativen Werten ein Schätzen von Wahrscheinlichkeiten mittels der Sigmoidfunktion ermöglicht. Ferner nimmt die Dimensionalität von Repräsentationen monoton ab. Betrachten wir ein lokales MLP im Modell von A. Poznanski und L. Wolf, so wird deutlich, dass für jedes n-Gram ein Featurevektor der Dimension 2048 aufgebaut wird, auf Basis dessen das n-Gram geschätzt wird. Das Vorgehen des MLP ist hinsichtlich einer Anwendung von CCA dahingehend verschieden, dass im Modell von A. Poznanski und L. Wolf Bildinformationen, über den Übergang der ersten zur zweiten Schicht eines jeden lokalen MLP, vervielfältigt in einem Vektor berücksichtigt werden können, um dann über eine Dimensionsreduktion und CCA einen reduzierten Attributvektor zu erhalten. Im Modell des PHOCNet wird die Dimensionsreduktion direkt über das MLP realisiert. Wir gehen dabei davon aus, dass die Dimensionsreduktion des MLP im PHOCNet ausreichend gut realisiert wird. Aufgrund dieser Argumentierung wird CCA auf den Ausgaben des PHOCNet und den PHOC der Wörter im Lexikon angewandt.

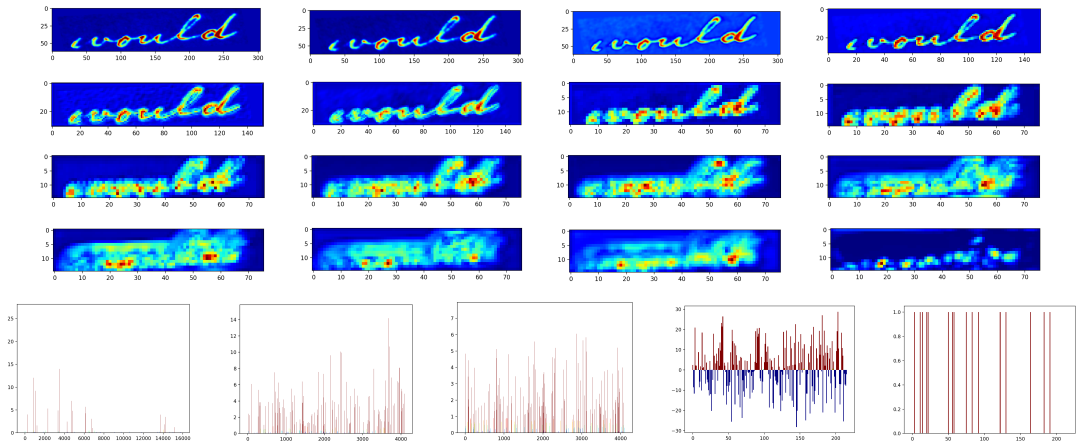

Abbildung 4.4.1: Die Ausgaben einer jeden Schicht für die Forwardpropagation eines Wortabbildes im PHOCNet.

5 EXPERIMENTE

Für die Evaluierung werden verschiedene Konfigurationen des PHOC*W*Rec Modells untersucht. Konkret werden für jeden Datensatz Alphabete, Augmentierungsvefahren, Skalierung der Eingabebilder, Pretraining und verschiedene Klassifikatoren evaluiert. Dabei wird sukzessiv vorgegangen und jeweils das beste Modell für den nächsten Evaluierungsschritt als Basis verwendet. Initial basiert das Modell auf einer gleichverteilten Augmentierung der Wortklassen, der originalen Skalierung der Eingabebilder, keinem Pretraining und einer nächster Nachbar Suche mit der Kosinus-Ähnlichkeit als Metrik für die Klassifikation.

5.1 DATENSÄTZE

Das PHOC*W*Rec Modell wird auf drei Datensätzen evaluiert. Die Datensätze weisen verschiedene Charakteristika hinsichtlich dem Umfang ihrer Daten, sowie den enthaltenen Wortabbildern und deren Transkriptionen auf. Im folgenden werden die Datensätze vorgestellt.

5.1.1 *George Washington*

Der George Washington Datensatz [RM07] umfasst isolierte Bilder zu 4.860 englische Wörter. Es handelt sich um einen historischen Datensatz, der von einem einzigen Schreiber verfasst wurde. Dies macht ihn hinsichtlich der Varianz von Handschriften homogen. Für den George-Washington Datensatz wurde von Almazán et al. Splits des Datensatz für eine Kreuzvalidierung veröffentlicht, die für die Evaluierung verwendet werden [AGFV14].

Das Lexikon des George Washington Datensatzes beläuft sich mit der Anzahl von Wortklassen der Trainings- und Testdaten auf 1471 Transkriptionen. Hinsichtlich dessen, dass der Datensatz vergleichsweise klein ist, wird ein PHOCNet für 100000 Iterationen auf den Trainingsdaten trainiert.

Abbildung 5.1.1: Variationen von Wortabbildern des George Washington Datensatzes. [RM07]

5.1.2 *IAM*

Ein im Bereich der Dokumentenanalyse verbreiteter Datensatz ist der IAM Datensatz [MB02]. Der IAM Datensatz umfasst 115320 segmentierte und transkribierte Wortabbilder von handgeschriebenem Text. Für den Datensatz wurden englische Texte von 657 Schreibern gesammelt, damit ist er vergleichsweise heterogen. Ein Vorteil des Datensatzes ist, dass offizielle Trainings-, Test- und Validierungsmengen veröffentlicht wurden. Dies erleichtert den Vergleich mit anderen Arbeiten.

Hinsichtlich unseres Ansatzes beläuft sich die Anzahl von Wortklassen im Lexikon auf 9474 Transkriptionen aus Test- und Trainingsmengen. Da der IAM Datensatz vergleichsweise groß ist, wird das PHOCNet für 200000 Iterationen auf diesem Datensatz trainiert.

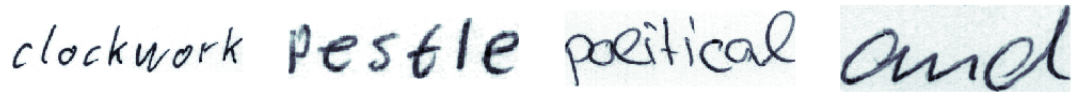

Abbildung 5.1.2: Variationen von Wortabbildern des IAM Datensatzes. [MB02]

5.1.3 *RIMES*

Der RIMES Datensatz [ABC+06] umfasst handgeschriebene Briefe von 1300 Schreibern zu 50000 transkribierten Bildern von französischen Wörtern und ist damit ein vergleichsweise heterogener Datensatz. Da französische Buchstaben verwendet wer-

Abbildung 5.1.3: Variationen von Wortabbildern des RIMES Datensatzes. [ABC+06]

den, bilden wir mit α alle Buchstaben mit Akzentzeichen auf die nächst ähnlichen Buchstaben des englischen Alphabets wie folgt ab:

$$\alpha(x) = \begin{cases} a & , x \in \{\grave{a}, \hat{a}\} \\ c & , x \in \{\c{c}\} \\ e & , x \in \{\grave{e}, \acute{e}, \hat{e}, \ddot{e}\} \\ i & , x \in \{\hat{i}, \ddot{i}\} \\ o & , x \in \{\hat{o}\} \\ u & , x \in \{\hat{u}, \grave{u}, \ddot{u}\} \end{cases} \tag{5.1.1}$$

Ferner sind für den RIMES Datensatz offizielle Splits für Trainings- und Testmengen bekannt.

Hinsichtlich des Lexikons umfasst der RIMES Datensatz 4928 Wortklassen. Ferner wird das PHOCNet auf dem RIMES Datensatz für 200000 Iterationen trainiert, da dieser vergleichsweise groß ist.

5.1.4 *HW-Synth*

Der IIIT-HWS, oder auch HW-Synth, Datensatz ist ein synthetisch generierter Datensatz von Wortabbildern und ihren Transkriptionen. Vollständig umfasst dieser 90 Millionen Wortabbilder, mit 90000 Wortklassen, und 100 Wortabbildern je Wortklasse.

Die Wortabbilder des HW-Synth Datensatzes werden aus ungefähr 750 synthetischen Fonts für Handschriften generiert, siehe Abbildung 5.1.4. Der HW-Synth Datensatz wird sowohl in einer 90K Version, die sämtliche Wortklassen umfasst, als auch in einer 10K Version, die nur eine Teilmenge von 10000 Wortklassen und dementsprechend 10 Millionen Wortabbildern umfasst, bereit gestellt. Im Rahmen dieser Arbeit wird nur die

Abbildung 5.1.4: Variationen von Wortabbildern des HW-Synth Datensatzes. [KJ16]

Teilmenge des HW-Synth Datensatzes mit 10000 Wortklassen und dementsprechend 10 Millionen Wortabbildern für das Vortrainieren des PHOCNet verwendet.

Hinsichtlich des Trainings auf HW-Synth verwenden wir einen Mini-Batch der Größe 32 und einer initialen Learning-Rate von $0,00005$ für eine Anzahl von 100000 Iterationen. Ferner wird nicht augmentiert, da die Menge von Wortabbildern des Datensatzes die $3,2$ Millionen Bilder des Trainings bereits überschreitet und der Datensatz lediglich zum Pretraining verwendet wird.

5.2 EVALUIERUNG

Für die Evaluierung der Modelle verwenden wir die *Word-Error-Rate* (WER) und die *Character-Error-Rate* (CER) als Gütemaß. Im Folgenden benennen wir mit $T = \{t_1, \ldots, t_N\}$ die Menge der Transkriptionen und mit $W = \{w_1, \ldots, w_N\}$ die Menge der geschätzten Wortklassen für Eingabebilder.

Gegeben eines geschätzten Wortes $w \in W$ und der Transkription $t \in T$, für ein Eingabebild, beschreibt der Word-Error (WE), ob zwei Wörter gleich sind. Die Word-Error-Rate beschreibt dementsprechend, wie viel Prozent aller Wörter im Testdatensatz falsch klasssifiziert wurden. Es folgen die Formeln für WE und WER

$$WE = \sum_{i=1}^{N} [\![t_i = w_i]\!] \tag{5.2.1}$$

$$WER = \frac{WE}{N}, \tag{5.2.2}$$

wobei die Iversons Klammernotation einen booleschen Wert auf 1 für wahr und 0 für falsch abbildet [Knu92].

Der Character-Error (CE) beschreibt wie viele Buchstaben in w minimal hinzugefügt, ausgetauscht oder gelöscht werden müssten um t zu erhalten, relativ zur Wortlänge der gewünschten Transkription. Definieren wir dazu $e_{ins}, e_{sub}, e_{del}$ als die Funktionen,

die zwei Wörter auf die minimale Anzahl von hinzugefügten, ausgetauschten und gelöschten Buchstaben abbildet, um die Formeln für CE und CER zu definieren:

$$CE = \sum_{i=1}^{N} \frac{e_{ins}(t_i, w_i) + e_{sub}(t_i, w_i) + e_{del}(t_i, w_i)}{|t_i|} \tag{5.2.3}$$

$$CER = \frac{CE}{N} \tag{5.2.4}$$

5.2.1 *Alphabete der PHOC Repräsentation*

Hinsichtlich des Alphabets, auf Basis dessen die HOC generiert werden, betrachten wir sowohl ein Alphabet, das sämtliche Buchstaben des Lexikons enthält, als auch Kombinationen von Buchstabensätzen. Im folgenden referenzieren wir mit L zu dem Buchstabensatz der Kleinbuchstaben, mit U zu dem Buchstabensatz der Großbuchstaben, mit D zu der Menge von Ziffern und mit P zu der Menge von Satzzeichen, wie in Abbildung 4.3.2 dargestellt. Werden mehrere Buchstabensätze, wie z.B. Klein- und Großbuchstaben, verwendet, bezeichnet LU, dass auf beide Buchstabensätze abgebildet wird. Im Falle, dass keine Großbuchstaben im Buchstabensatz enthalten sind werden auch alle Wortklassen des Lexikons dementsprechend angepasst. Alle Wortklassen bleiben weiterhin erhalten, indem sie auf eine äquivalente Wortklasse mit Kleinbuchstaben abgebildet werden. Diese Regel ist einzigartig, da hier das Lexikon modifiziert wird. Sollten keine Ziffern oder Satzzeichen im Buchstabensatz enthalten sein wird das Lexikon nicht modifiziert. Zu Alphabeten, die exakt die Buchstaben des ursprünglichen Lexikons eines Datensatzes enthalten, wird im Folgenden als *vollständig* referenziert. Dies heißt nicht, dass sie inhärent optimal gewählt sind, sondern lediglich, dass ihr Buchstabensatz genau der Menge von Buchstaben des Lexikons vom Datensatz, unter Berücksichtigung der Großbuchstaben, entsprechen. Solche vollständigen Alphabete sind in Abbildung 5.2.1 gelistet. Interessant hierbei ist, dass das vollständige Alphabet des George Washington Datensatz dem Buchstabensatz

Datensatz	Alphabet
IAM	!"#& '()*+,-./0123456789:;?abcdefghijklmnopqrstuvwxyzABC...
GW	0123456789abcdefghijklmnopqrstuvwxyz
RIMES	% '-/0123456789?abcdefghijklmnopqrstuvwxyzABC...

Abbildung 5.2.1: Auf die Datensätze abgestimmte Alphabete.

LD entspricht. Dies ist für diesen Teil der Evaluierung verschiedener Konfigurationen deshalb von Bedeutung, da wir untersuchen möchten, wie stark Ungenutzte Attribute die Klassifikation beeinträchtigen und ob ein vereinfachtes Alphabet mit ausschließlich Kleinbuchstaben die Klassifikation erleichtert.

Für jeden Datensatz wurden Instanzen des PHOCNet für die verschiedenen Buchstabenkombinationen trainiert. Es wurde kein Buchstabensatz ohne Ziffern und Buchstaben in Betracht gezogen, da Wortabbilder, die nur Satzzeichen enthalten sehr selten und je nach Datensatz nicht existent sind. Die Klassifikation erfolgt durch eine nächster Nachbar Suche mit Kosinus-Ähnlichkeit, wie im Unterkapitel 4.4 beschrieben. Es wurde mit gleichverteilten Wortklassen augmentiert und die Eingabebilder entsprachen ihrer originalen Skalierung. Resultierende Fehlerraten sind der Tabelle 5.2.1 zu entnehmen.

Tabelle 5.2.1: Fehlerraten für verschiedene Alphabete. Alle Modelle verwendet eine nächster Nachbar Suche als Klassifikator, eine gleichverteilte Augmentierung, die originale Skalierung der Eingabebilder und kein Pretraining.

Datensatz	Alphabet	WER	CER
George Washington	**LD / vollständig**	**8,66%**	**5,08%**
George Washington	LDP	8,71%	5,11%
RIMES	vollständig	16,71%	9,87%
RIMES	LD	18,08%	**9,32%**
RIMES	LUD	24,06%	11,22%
RIMES	**LDP**	**16,10%**	9,47%
RIMES	LUDP	24,61%	11,62%
IAM	vollständig	34,67%	48,40%
IAM	LD	35,17%	49,16%
IAM	LUD	42,81%	86,98%
IAM	**LDP**	**29,95%**	**35,85%**
IAM	LUDP	38,84%	41,05%

Aus der Evaluierung verschiedener Alphabete ist ersichtlich, dass generell ein vollständiges Alphabet zu besseren Ergebnissen führt, als ein Alphabet, das über das vollständige Alphabet hinaus Buchstaben enthält. Die besten Ergebnisse für

Datensätze, die Groß- und Kleinschreibung berücksichtigen wurden jedoch durch den Buchstabensatz LDP als Alphabet erzielt. Daraus kann geschlossen werden, dass die Abbildung auf Kleinbuchstaben die Klassifizierung und gegebenenfalls auch das Training erleichtern kann.

5.2.2 *Augmentierung von Trainingsdaten*

Für die Augmentierung der Trainingsdaten wurden, wie in 4.3.5 erörtert, affine Transformationen verwendet. Ziel der Augmentierung ist es weitere Trainingsdaten auf Basis der bekannten Daten zu generieren. Im Rahmen dieser Arbeit werden die Trainingsdatensätze durch Augmentierung um so viele Wortabbilder erweitert, dass sie insgesamt eine halbe Millionen Wortabbilder umfassen. Hierbei muss entschieden werden welche Wortklassen wie oft vorkommen sollen, da dies unmittelbar die Verteilung von Attributen und deren räumliche Verteilung beeinflusst.

Werden Wortklassen so verteilt, dass ihre relative Häufigkeit gewahrt wird, so erhält man einen *ungleich* verteilten, augmentierten Datensatz. Im Kontrast dazu steht ein *gleich* verteilter, augmentierter Datensatz, bei dem alle Wortklassen die selbe Häufigkeit besitzen. Training mit einem ungleich verteilten Datensatz kann in herkömmlichen neuronalen Netzen, die direkt Klassifizieren, dazu führen, dass Klassen, die im Training häufiger vorkommen, präferiert werden. Das PHOCNet verfolgt mit dem Schätzen von Attributen einen anderen Task als direkt klassifizierende neuronale Netze. Ein PHOC encodiert Buchstaben und deren räumliche Verteilung. Damit die Daten für das PHOCNet hinsichtlich des Tasks, das respektive PHOC zu schätzen, gleich verteilt wären, müssten Buchstaben und deren räumliche Verteilung gemäß einer PHOC Repräsentation gleich verteilt sein. Eine Solche Verteilung mittels Wortklassen, die nicht ausgelassen oder modifiziert werden sollen, zu generieren ist jedoch nicht offensichtlich, wenn lösbar. Eine ungleich verteilte Augmentierung der Wortklassen induziert jedoch eine ungleich verteilte Augmentierung der Attribute. Die relative Häufigkeit von Buchstaben unter Berücksichtigung ihrer räumlichen Verteilung bleibt gewahrt.

Um zu untersuchen welchen Unterschied die Verteilung der Wortklassen ausmacht wurden auf allen relevanten Datensätzen Modelle mit gleich und ungleich verteilt augmentierten Wortklassen trainiert. Dabei wurden die Alphabete, die zuvor zu den besten Resultaten der WER führten als Basis für die Modelle verwendet, die Fehlerraten sind der Tabelle 5.2.2 zu entnehmen.

Insbesondere auf den vergleichsweise heterogenen Datensätzen IAM und RIMES führte eine ungleich verteilte Augmentierung zu signifikanten Verbesserungen. In der

Tabelle 5.2.2: Fehlerraten für gleich- und ungleichverteilte Augmentierung der Trainingsdaten.

Datensatz	Verteilung	WER	CER
George Washington	gleich	8,66%	5,08%
George Washington	**ungleich**	**7,14%**	**3,89%**
RIMES	gleich	16,1%	9,47%
RIMES	**ungleich**	**8,19%**	**3,65**
IAM	gleich	29,95%	35,85%
IAM	**ungleich**	**15,71%**	**9,97%**

initialen Veröffnetlichung des PHOCNet [SF18] wurde von S. Sudhold und G. A. Fink eine gleich verteilte Augmentierung der Wortklassen vorgeschlagen, was zu guten Ergebnissen führte. Die Verbesserung durch eine ungleich verteilte Augmentierung kann dadurch begründet werden, dass das PHOCNet, über ungleich verteiltes Augmentieren der Trainingsdaten, die Verteilung der Attribute mit lernt. Vergleichbar zu einer Art Sprachannahme auf Basis der Trainingsdaten werden die Attribute der PHOC hinsichtlich ihrer Frequenz gewichtet und bestimmt. Hierbei sollte aber berücksichtigt werden, dass eine ungleich verteilte Augmentierung voraus setzt, dass die Attribute der Testdaten vergleichbar zu derer, der Trainingsdaten verteilt sind. Diese Annahme kann nicht verallgemeinert getroffen werden. Dementsprechend kann die Erkenntnis, dass eine ungleich verteilte Augmentierung zu einer Verbesserung führt nur für Datensätze mit ähnlich verteilten Attributen in den Test- und Traingsdaten aus der Evaluierung geschlossen werden.

5.2.3 *Skalierung der Eingabebilder*

Im Folgenden referenzieren wir zu der Höhe eines Eingabebildes mit h und zu dessen Breite mit w, sowie zur Höhe eines modifizierten Eingabebildes mit h' und dessen Breite mit w' . Für die Skalierung der Eingabebilder werden neben der unveränderten, originalen Skalierung auch Skalierungen mit einer Höhe $h' = 50$ betrachtet. Da es möglich ist, unter Wahrung der Relation von Breite und Höhe des Eingabebildes, Bildinformation zu reduzieren, ohne das Eingabebild zu stauchen oder strecken, betrachten wir eine Skalierung der Breite zu $w' = \frac{w}{h} \cdot h'$. Diese Skalierung nennen wir $50,-$ Skalierung. Alternativ wird mit $w' = 100$ eine Skalierung betrachtet, die

Eingabebilder stauchen oder strecken kann. Diese Skalierung nennen wir 50, 100 Skalierung.

Es wurden Modelle mit den genannten Skalierungen trainiert. Für jedes Modell wurde der Datensatz ungleich augmentiert, da dies auf allen Datensätzen zu besseren Ergebnissen führte. Die Fehlerraten sind der Tabelle 5.2.3 zu entnehmen.

Tabelle 5.2.3: Fehlerraten für verschiedene Skalierungsansätze der Eingabebilder.

Datensatz	Skalierung	WER	CER
George Washington	**original**	**7,14%**	**3,89%**
George Washington	50, –	8,85%	5,45%
George Washington	50, 100	8,25%	4,65%
RIMES	**original**	**8,19%**	**3,65%**
RIMES	50, –	10,67%	5,74%
RIMES	50, 100	8,23%	3,91%
IAM	original	15,71%	9,97%
IAM	**50, –**	**15,10 %**	**9,52 %**
IAM	50, 100	15,47%	10,09%

Die Skalierung der Eingabebilder führt nur auf dem IAM Datensatz zu Verbesserungen der Fehlerraten. Dies könnte darin begründet werden, dass die Wortabbilder des IAM Datensatz vergleichsweise groß sind. Die durch die 50, – oder 50, 100 Skalierung reduzierte Anzahl von Parametern könnte sich vorteilhaft auf das PHOCNet auswirken. In Betracht der Ergebnisse auf den anderen Datensätzen kann jedoch keine Aussage darüber getroffen werden, ob eine 50, 100 oder 50, – Skalierung allgemein besser, als die originale Skalierung geeignet ist. Für die Anwendung einer 50, 100 oder 50, – Skalierung auf die Eingabebilder, ist aus den Experimenten keine klare Tendenz hinsichtlich der Fehlerraten ersichtlich.

5.2.4 *Vortrainierte Modelle*

Eine Möglichkeit ein neuronales Netz für einen Datensatz zu initialisieren ist mittels der Parameter einer Instanz, die vortrainiert wurde. Hinsichtlich des Task der Attributerkennung auf Basis eines PHOC und einer unbekannten Verteilung von Attributen

im Datensatz, ist es wünschenswert ein PHOCNet auf einem Datensatz mit möglichst vielen verschiedenen Wortklassen vorzutrainieren. Von einer Vielzahl von Worklassen erhofft man sich bereits die Erkennung relevanter Attribute angelernt zu haben.

Hinsichtlich der Vielfalt an Wortklassen bietet sich der HW-Synth Datensatz für das Pretraining an. Ferner enthält der HW-Synth Datensatz Wortabbilder für Konkatenationen von gleichen Buchstaben, wie z.B. *"zzzzz"*. Dies sollte implizieren, dass für nahezu alle Attribute, des Attributsvektors zum PHOC, Trainingsdaten vorhanden sind. Da die Wortklassen des HW-Synth Datensatzes für den Zweck eines Pretrainings angemessen verteilt sind, werden die Trainingsdaten für die vortrainierte Instanz nicht erweitert. Ferner werden keine affinen Transformationen an besagten Trainingsdaten vorgenommen und damit die Daten nicht augmentiert. Das Pretraining erfolgt wie in 5.1.4 beschrieben.

Für alle Datensätze wurden PHOCNet Instanzen auf dem HW-Synth Datensatz mit respektiver Konfiguration vortrainiert. Dazu wurden für RIMES und George Washington die originalen Skalierungen beibehalten, da keine Verbesserung unter modifizierten Skalierungen ersichtlich waren. Für IAM wurde wiederum die $50,-$ Skalierung angewandt, da diese zu den besten Ergebnissen führte. Fehlerraten für die vortrainierten Modelle sind der Tabelle 5.2.4 zu entnehmen.

Tabelle 5.2.4: Fehlerraten vortrainierter Modelle.

Datensatz	vortrainiert	WER	CER
George Washington	**ja**	**3,75%**	**1,94%**
George Washington	nein	7,14%	3,89%
RIMES	**ja**	**6,00%**	**2,62%**
RIMES	nein	8,19%	3,65%
IAM	**ja**	**12,17%**	**7,65%**
IAM	nein	15,71%	9,97%

Das Pretraining führt auf allen drei Datensätzen zu einer Verbesserung. Es kann gefolgert werden, dass das PHOCNet von einem Pretraining auf HW-Synth profitiert. Hinsichtlich dessen, dass zuvor bereits alle anderen Parameter für das PHOCNet evaluiert wurden ist die aus dem Pretraining resultierende Verbesserung der WER von 2,19 bis 3,54 Prozent und der CER von 1,03 bis 2,32 Prozent signifikant.

5.2.5 *Klassifikatoren*

Hinsichtlich der Klassifikatoren betrachten wir einen nächster Nachbar Klassifikator mit verschiedenen Distanzmetriken, das Probabilistische Rückgabemodell nach Rusakov et al. [RRMF18] und einen Klassifikator der eine nächster Nachbarsuche in einem durch CCA ermittelten Unterraum durchführt.

Für die Klassifikation mittels einer nächster Nachbar Suche gilt es eine Metrik festzulegen. Die Distanzmetrik ist für diesen Klassifikator der einzige Hyperparameter. Im Rahmen dieser Arbeit wird die Kosinus-Ähnlichkeit:

$$\cos(\alpha) = \frac{a^T \hat{a}}{||a||_2 ||\hat{a}||_2} \tag{5.2.5}$$

und die euklidische Distanz:

$$\mathrm{dist}_{euc}(a, \hat{a}) = \sqrt{\sum_{i=1}^{D} (a_i \cdot \hat{a}_i)^2} \tag{5.2.6}$$

als Metrik, gegeben zwei Attributvektoren $a, \hat{a}$, in Erwägung gezogen.

Tabelle 5.2.5: Fehlerraten für verschiedene Metriken eines Klassifikators, der auf einer nächster Nachbar Suche beruht.

Datensatz	Metrik	WER	CER
George Washington	**cos**	**3,75%**	**1,94%**
George Washington	euklidisch	4,08%	2,05%
RIMES	**cos**	**6,00%**	2,62%
RIMES	euklidisch	6,25 %	**2,55%**
IAM	**cos**	**12,17%**	7,65%
IAM	euklidisch	12,78%	**7,43%**

Da in der CCA die maximale kanonische Korrelation $\cos(\Theta)$ über den Winkel Θ ermittelt wird, in anderen Worten der Unterraum so aufgespannt wird, dass nach CCA ähnliche Attributvektoren im Unterraum eine hohe Kosinus-Ähnlichkeit aufweisen, wird auch nur die Kosinus-Ähnlichkeit für die nächster Nachbar Suche im Unterraum verwendet. Ferner wurde die Kosinus-Ähnlichkeit auch von Poznanski und Wolf in [PW16] als Metrik für eine nächster Nachbar Suche vorgeschlagen.

Für eine nächster Nachbar Suche zwischen den nativen Attributvektoren lässt sich eine solche Aussage nicht treffen. In [SF18] wurde die Kosinus-Ähnlichkeit als Metrik für die Sortierung der Rückgabeliste verwendet. Jedoch auch erwähnt, dass das Modell mit einer Kosinus- und einer euklidischen Fehlerfunktion trainiert werden kann. Für die nächster Nachbar-Suche werden wir im folgenden, wie auch von S. Sudhold und G. A. Fink vorgeschalgen die Kosinus-Ähnlichkeit verwenden. Um diese Wahl noch einmal zu fundieren wurden auf allen Datensätzen nächster Nachbar Klassifikatoren mit besagten Metriken evaluiert. In der Tabelle 5.2.5 sind die Fehlerraten abgebildet. Da auch hier die Kosinus-Ähnlichkeit zu einer niedrigeren *WER* auf allen Datensätzen führte, wird diese im Folgenden als die Metrik der nächster Nachbar Suche festgelegt.

Der CCA-Klassifikator besitzt noch weitere Hyperparameter. Da eine regularisierte CCA verwendet wird, müssen der Regularisierungsparameter r und die Anzahl der resultierenden Dimensionen im Unterraum d passend gewählt werden. Um dies zu realisieren verwenden wir eine *Cross Validation*, oder auch Kreuzvalidierung, über der Menge von möglichen Parametern für die Regularisierung $R = \{1, 10, 10^2, 10^3, 10^4\}$ und möglichen Parametern der resultierenden Dimension $D = \{16, 32, 64, 128, 256\}$. Für die Kreuzvalidierung werden unsere Trainingsdaten randomisiert in vier Quartale mit je 25% der Trainingsdaten eingeteilt. Es wird auf jedem Quartal genau einmal evaluiert, siehe Abbildung 5.2.2.

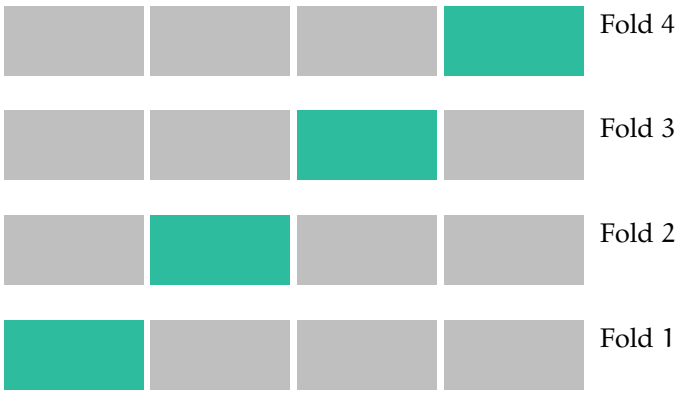

Abbildung 5.2.2: Bei der Kreuzvalidierung wird ein Datenbestand in k gleichgroße Folds aufgeteilt. Ihm Rahmen der Validierung dient jeder Fold einmal als Testdatensatz während auf den restlichen Daten trainiert wird. Abgebildet ist die Aufteilung von Trainingsdaten (grau) und Testdaten (grün) in den jeweiligen Folds für $k = 4$.

Dazu wird zu jeder Konfiguration r, d aus $R \times D$ ein CCA-Klassifikator auf den jeweils drei übrigen Quartalen angelernt. Wir ermitteln in jedem Evaluierungsschritt die Wortfehler und summieren diese für jede Konfiguration respektiv. Die Resultate einer solchen Kreuzvalidierung sind in Abbildung 5.2.3 visualisiert. Am Ende wird eine CCA auf den gesamten Trainingsdaten, mit der Konfiguration, die im Rahmen der Kreuzvalidierung die geringsten Wortfehler aufwies, angelernt. Diese CCA wird für den CCA-Klassifikator genutzt.

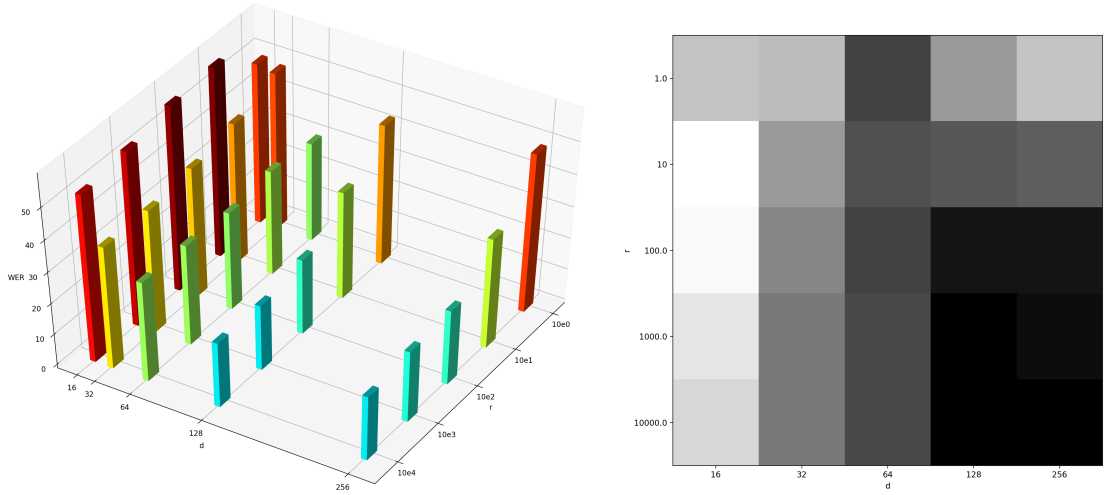

Abbildung 5.2.3: Summierte Wortfehler über alle Faltungen der Crossvalidierung für den Regularisierungsparameter r und der Dimensionalität d der CCA. Evaluiert wurde auf dem George-Washington Datensatz. Im Balkendiagramm (links) ist die WER im Verhältnis zu verschiedenen Konfigurationen für d und r abgebildet, auf der Karte (rechts) impliziert schwarz eine niedrige und weiß eine hohen WER.

Da, wie in 3.3 erörtert das Probablistische Modell nach Rusakov et al. [RRMF18] direkt auf das PHOCNet anwendbar ist und keine weiteren Hyperparameter birgt, müssen keine Konfigurationen dieses Klassifikators betrachtet werden. Ferner könnte man sogar den Vergleich des Probabilischen Modells mit einer nächster Nachbar Suche führen. Hierbei würde die nächster Nachbar Suche auf Basis der PRM-Scores operieren und größere Werte eine höhere Ähnlichkeit implizieren. Die Fehlerraten aller Klassifikatoren sind der Tabelle 5.2.6 zu entnehmen.

Aus der Evaluierung der Klassifikatoren kann entnommen werden, dass die Klassifikation über eine nächster Nachbar Suche, mit der Kosinus-Ähnlichkeit als Metrik, auch im Vergleich zu komplexeren Klassifikationsansätzen solide Ergebnisse liefert.

Tabelle 5.2.6: Fehlerraten für verschiedene Klassifikationsansätze.

Datensatz	Klassifikator	WER	CER
George Washington	**cos**	**3,75%**	**1,94%**
George Washington	PRM	3,91%	2,01%
George Washington	CCA	17,24%	8,52%
RIMES	**cos**	**6,00%**	2,62%
RIMES	PRM	6,02	**2,48%**
RIMES	CCA	9,23%	3,89%
IAM	cos	12,17%	7,65%
IAM	**PRM**	**11,86%**	**6,83%**
IAM	CCA	22,35%	12,00%

Ferner scheint das Probabilistische Modell bei vergleichsweise heterogenen Datensätzen besonders gut zu funktionieren. Dies kann darin begründet werden, dass für heterogene Datensätze die geschätzten PHOC gegenüber den eigentlichen PHOC der Transkriptionen eine höhere Varianz aufweisen können, als für homogene Datensätze. Dadurch, dass das Probabilistische Modell auch die Gegenwahrscheinlichkeiten berücksichtigt, kann es die geschätzten Attributvektoren dann besser interpretieren. Gleichen die geschätzten PHOC wiederum den PHOC der Transkriptionen sehr stark, so resultieren die PRM-Scores in leicht verrauschten Werten, da dann mit den Gegenwahrscheinlichkeiten Informationen berücksichtigt werden, die vernachlässigbar wären. Dies könnte begründen warum das Probabilistische Modell auf dem George Washington Datensatz schlechter abschneidet, als eine nächster Nachbar Suche mit Kosinus-Ähnlichkeit. Des weiteren resultierte das in dieser Arbeit vorgestellte Klassifikationsverfahren über CCA zu schlechteren Ergebnissen. Dies kann daran liegen, dass die vorgestellte Kreuzvalidierung gegebenenfalls nicht optimal gewählt ist. Wie auch Abbildung 5.2.3 zu entnehmen ist, sind etwa die Wortfehler für den George Washington Datensatz für viele Konfigurationen sehr nahe beieinander. Dies kann in der sehr guten Performanz des PHOCNet auf den Trainingsdaten begründet werden. Folglich ist es unklar, ob durch die Kreuzvalidierung die beste Konfiguration der CCA bestimmt wird, oder ob die Kreuzvalidierung ein Overfitting auf den Trainingsdaten bewirkt. Ferner könnte die Eingabe der CCA mit den geschätzten PHOC

und den PHOC der Transkriptionen nicht optimal gewählt sein. Gerade wenn auf den Trainingsdaten die PHOC mit hoher Genauigkeit approximiert werden, lernt die CCA im Wesentlichen die PHOC-Repräsentationen gegen sich selbst. Sollten auf den Testdaten die PHOC dann weniger genau bestimmt werden, könnte dies zu einer Beeinträchtigung der Klassifikation im Unterraum führen.

5.3 RESULTATE UND VERGLEICH

Aus der Evaluierung kann geschlossen werden, dass Alphabete ohne Großbuchstaben die Klassifizierung erleichtern. Ferner führten eine ungleich verteilte Augmentierung und auf HW-Synth vortrainierte Modelle für alle Datensätze zu Verbesserungen. Auf Basis dieser Arbeit kann gefolgert werden, dass diese Verfahren allgemein zu einer Verbesserung für die Optimierung des PHOCNet führen.

Hinsichtlich der Skalierung und der Klassifikatoren kann keine allgemeine Aussage getroffen werden. Während normalisierte Skalierungen der Eingabe bei IAM eine Verbesserung bewirkten, konnte dies nicht auf den anderen Datensätzen festgestellt werden. Ähnlich verhielt es sich mit dem Probabilistischen Modell. Hinsichtlich des Probabilistischen Modells ließ sich feststellen, dass es besonders gut auf heterogenen Datensätzen angewandt werden kann, jedoch nicht für alle Datensätze besser geeignet ist. Zudem konnte festgestellt werden, dass der in dieser Arbeit vorgestellte Ansatz einer Klassifikation über CCA nicht hinreichend gut funktionierte, wie auch im Fazit 6 diskutiert wird.

Abschließend werden die besten Modelle je Datensatz mit anderen Arbeiten verglichen. Die genaue Konfiguration der Modelle kann aus Tabelle 5.3.1 abgelesen werden.

Tabelle 5.3.1: Konfigurationen von PHOCWRec auf den verschiedenen Datensätzen.

Modell Name	Alphabet	Augmentierung	Skalierung	Pretraining	Klassifikator
PHOCWRec_{GW}	LD	ungleich	original	ja	cos
PHOCWRec_{RIMES}	LDP	ungleich	original	ja	cos
PHOCWRec_{IAM}	LDP	ungleich	50, –	ja	PRM

Die Ansätze mit denen PHOCWRec verglichen wird umfassen das CNN-n-gram Modell [PW16] hybride Ansätze für HMMs und LSTMs, sowie ein state of the Art LSTM Modell. Für die hybriden Ansätze wird mit einem Modell der RWTH [DKN14], einem hybriden Ansatz von einem LSTM-RNN und einem HMM für die Evaluierung,

sowie mit dem ConvNN-GHMM [BNK13] ein Tandem Ansatz eines CNN mit einem *gaussian* HMM betrachtet. Fehlerraten sind der Tabelle 5.3.2 zu entnehmen.

Tabelle 5.3.2: Vergleich der Modelle hinischtlich der Fehlerraten WER und CER. Für alle Modelle werden nur Ergebnisse Lexikon basierter Ansätze betrachtet, für die auch Wortklassen der Testdaten bekannt sind.

Modell	Datensatz	WER	CER
PHOCWRec$_{GW}$	**George Washington**	**3,75%**	**1,94%**
Dutta et al. [DKMJ18]	George Washington	12,59%	3,81%
PHOCWRec$_{RIMES}$	RIMES	6,00%	2,62%
CNN-n-gram [PW16]	RIMES	3,90%	1,90%
Dutta et al. [DKMJ18]	**RIMES**	**1,86%**	**0,65%**
RWTH [DKN14]	RIMES	12,9%	4,3%
ConvNN-GHMM tandem +CD + MMI WR2 [BNK13]	RIMES	7,9%	-
PHOCWRec$_{IAM}$	IAM	12,17%	7,65%
CNN-n-gram [PW16]	IAM	6,45%	3,44%
Dutta et al. [DKMJ18]	**IAM**	**4,80%**	**2,52%**
RWTH [DKN14]	IAM	12,2%	4,7%
ConvNN-GHMM tandem +CD [BNK13]	IAM	20,5%	-

Im Vergleich zu anderen Modellen schneidet das PHOCWRec Modell auf den aufgeführten Datensätzen besser ab, als das Modell der RWTH [DKN14] und der Tandem Ansatz von [BNK13]. Hinsichtlich des ähnlichen Ansatzes von A. Poznanski und L. Wolf [PW16] schneidet das PHOCWRec Modell schlechter ab. Dies könnte in der besseren Realisierung der CCA von CNN-n-gram begründet sein. Interessant ist, dass PHOCWRec auf dem George Washington Datensatz wesentlich bessere Ergebnisse erzielt, als das state of the art Modell von Dutta et al. [DKMJ18]. Dies wurde aber auch von Dutta et al. in ihrer Veröffentlichung angemerkt und durch den geringen Umfang des Datensatzes, sowie den binarisierten Bildern und einer aus dem Kleinbuchstabenalphabet des George Washington Datensatzes resultierenden Unstimmigkeit im verwendeten Lexikon basierten Decodierungsverfahren begründet.

Hinsichtlich der Ergebnisse der CER ist zu erkennen, dass die aufgeführten Arbeiten bei vergleichbarer WER eine niedrigere CER aufwiesen. Es könnte vermutet werden, dass andere Arbeiten tendenziell Wörter mit kleinerem CE schätzen. Jedoch könnten die Unterschiede in der CER auch dadurch erklärt werden, dass gegebenenfalls in den aufgeführten Arbeiten die CER zusätzlich normiert wird. Die in 5.2.4 angegebene Formel der CER kann für den Fall eines falschen Wortes mit größerer Länge als die Transkription in einem CE über 100% resultieren. Würde die in dieser Arbeit verwendete Formel für die CER normalisiert werden, könnte dies die Fehlerrate verringern.

Obwohl in dieser Arbeit über das PHOCNet mit dem PHOCWRec ein vergleichsweise guter Ansatz für die Einzelworterkennung vorgestellt wurde, bleiben LSTM Ansätze, wie der von Dutta et al. [DKMJ18], der state of the art für heterogene Datensätze. Dennoch erscheinen die Fehlerraten auf dem homogenen George Washington Datensatz beachtlich, wenngleich für diesen Datensatz weitere Referenzwerte fehlen.

6 FAZIT

In dieser Arbeit wurde mit dem PHOCNet untersucht, wie ein Attribute CNN auf das Problem der Einzelworterkennung angewandt werden kann. Um das PHOCNet auf das Problem der Einzelworterkennung anzuwenden wurde ein Modell PHOC*W*Rec konstruiert, das auf Basis eines Lexikons von Wortklassen, gegeben eines geschätzten Attributvektors eine Wortklasse schätzt.

Eine Erkenntnis, die aus den Experimenten gewonnen werden kann ist, dass Alphabete ohne Großbuchstaben vorteilhaft für PHOC*W*Rec sind. Mögliche Gründe für die niedrigeren Fehlerraten sind, dass durch die Vereinfachung der Wortklassen etwa Buchstaben, die groß- und kleingeschrieben ähnlich sind auf das selbe Attribut abgebildet werden können und die resultierenden PHOC Vektoren eines solchen Alphabets eine kleinere Dimensionalität besitzen.

Des weiteren kann aus den Experimenten gefolgert werden, dass sich eine ungleich verteilte Augmentierung der Wortklassen positiv auf die Evaluierung der Testdaten auswirkt. Wie in 5.2.2 erörtert, ist es schwer, wenn lösbar, eine Gleichverteilung von Buchstaben in allen für das PHOC relevanten Teilstrings aus der festen, endlichen Wortmenge des Datensatzes zu generieren. Deshalb und insbesondere weil das PHOCNet als Attribute CNN nicht die Aufgabe einer direkten Klassifizierung verfolgt, induziert ein Trainingsdatensatz mit gleichverteilten Wortklassen keinen offensichtlich Vorteil hinsichtlich der Evaluierung auf den Testdaten. Eine ungleich verteilte Augmentierung wahrt die relative Häufigkeit eines jeden Attributs in den für das PHOC relevanten Teilstrings. Dadurch lernt das PHOCNet die Verteilung der Attribute im Trainingsdatensatz mit. Hinsichtlich der daraus resultierenden Gewichtung von Attributen im Kontext des PHOC, kann hierbei die ungleich verteilte Augmentierung mit einer Sprachannahme verglichen werden. Es ist möglich, dass wenn in einem für das PHOC relevanten Teilstring der Buchstabe *"l"* im Trainingsdatensatz infrequent vorkommt und das große i *"I"* frequent, das optimierte PHOCNet für das respektive HOC das große i präferiert. Werden die Resultate kritisch betrachtet, so implizieren wir mit einer ungleich verteilten Augmentierung, dass die relativen Häufigkeiten der Attribute in den Trainingsdaten derer der Testdaten entsprechen. Dies ist nicht zwingend der Fall und sollte diese Annahme nicht zutreffen ist kein Vorteil einer ungleich verteilten Augmentierung gegenüber einer gleichverteilten Augmentierung der Wortklassen ersichtlich.

Würde man annehmen, dass man auf einem Datensatz operiert, für den keine Aussage hinsichtlich der relativen Häufigkeit von Buchstaben hinsichtlich der für ein PHOC relevanten Teilstrings getroffen werden kann, so wäre eine Art Gleichverteilung der Attribute wünschenswert. Der *HW*-Synth Datensatz bietet dabei eine gute Approximation für eine verallgemeinerte Verteilung von Attributen. Dies zeigte sich auch in den Experimenten. Modelle, die auf *HW*-Synth vortrainiert wurden erzielten deutlich niedrigere Fehlerraten.

Hinsichtlich der Skalierung der Eingaben konnte keine klare Tendenz festgestellt werden. Während die originale Skalierung auf dem George Washington und RIMES Datensatz zu den besten Ergebnissen führte, resultierten die normalisierten Skalierungen $50, -$ und $50, 100$ auf dem IAM Datensatz in einer Verbesserung. Dies könnte daran liegen, dass die Wortabbilder des IAM Datensatzes vergleichsweise groß sind und durch eine reduzierte Eingabe mit weniger Parametern das PHOCNet effektiver optimiert werden kann. Dennoch ist aus den Experimenten nicht ersichtlich welche Vor- und Nachteile eine Skalierung der Eingabebilder hat.

Neben der Konfiguration von Attributvektor, Augmentierung und dem Pretraining stellte sich die Wahl eines geeigneten Klassifikators als wesentlichen Faktor für die Einzelworterkennung heraus. Während wir zuvor Aussagen für alle Datensätze treffen konnten, kann auf Basis der Experimente gefolgert werden, dass der Optimale Klassifikator von Datensatz zu Datensatz variieren kann. Die einfache nächster Nachbar Suche mittels einer geeigneten Distanzmetrik stellt sich als guter Indikator dafür heraus, wie exakt das PHOCNet die gesuchten Attributvektoren approximiert. Es werden keine Gegenwahrscheinlichkeiten berücksichtigt und die Dimensionalität der Attributvektoren nicht reduziert. Dieser Ansatz beruht darauf, dass die geschätzten Attributvektoren nahezu binär sind und den gesuchten PHOC beinahe gleichen. In diesem Fall ist die nächster Nachbar Suche mit einer einfachen Metrik, wie etwa der Kosinus-Ähnlichkeit, nicht nur ausreichend, sondern auch besser als das Probabilistische Modell, dessen Entscheidungsfunktion auf der Gegenwahrscheinlichkeit des geschätzten Attributvektors, beruht.

Das Probabilistische Modell wies jedoch deutlich bessere Werte auf, je heterogener der Datensatz und in Konsequenz weniger akkurat die Attributvektoren waren. Die Kosinus-Ähnlichkeit berücksichtigt nur die Wahrscheinlichkeit, dass ein Attribut vorhanden ist. Für weniger akkurat geschätze Attributvektoren resultiert dies in einem Informationsverlust, der durch die Berücksichtigung der Gegenwahrscheinlichkeit im Probabilistischen Modell kompensiert werden kann. Somit extrahiert das Probabilistische Modell mehr Informationen aus den Attributen des Attributvektors. Während dies für sehr exakte Attributvektoren in einem, im Vergleich zu einer nächster Nachbar

Suche mit der Kosinus-Ähnlichkeit, leicht verrauschten Wert resultiert, ermöglicht es weniger akkurat geschätzte Attributvektoren besser zu vergleichen.

Eine weitere Erkenntis, die aus der Arbeit gezogen werden kann, ist, dass das in dieser Arbeit vorgestellte Verfahren einer Klassifikation über CCA auf Attributvektoren, zu keinen guten Ergebnissen führte. Mögliche Gründe für die schlechten Resultate des vorgestellten Verfahrens sind ein starkes Overfitting auf den Trainingsdaten und im Falle einer zu geringen Dimensionalität, der daraus resultierende Informationsverlust. Für die Kreuzvalidierung wurde die WER als Gütemaß verwendet, da dieser als robust angenommen wurde. Für Datensätze wie den George Washington Datensatz, für den das verwendete PHOCNet bereits nahezu die PHOC der Transkriptionen der Eingabebilder ausgibt, kann es Vorkommen, dass mehrere Konfigurationen die selbe WER haben. In diesem Fall sind die Trainingsmengen für die CCA nahezu identisch. Dies führt zu einer schlecht fundierten Wahl zwischen den, auf Basis der Kreuzvalidierung, besten Konfigurationen für die CCA. Daraus lässt sich schließen, dass dieser Ansatz einer Kreuzvalidierung für die Konfiguration einer CCA suboptimal gewählt ist. Ferner ist das PHOCNet, im Gegensatz zum neuronalen Netz, dass in CNN-n-Gram [PW16] vorgestellt wurde, nicht dafür konstruiert worden Neural Codes, in vergleichbarer Weise, zu generieren. Aufgrund dessen wurden keine Ausgaben versteckter Schichten für die CCA in Betracht gezogen. Hinsichtlich neuer Ansätze für die Einbindung von CCA in einem zu PHOCWRec ähnlichem Modell, könnten also noch versteckte Schichten, mit höherer Dimensionalität anstelle der Attributvektoren in Betracht gezogen werden. Ferner könnte versucht werden die CCA mittels der Ausgaben eines anderen Klassifikators, wie z.B. das Probabilistische Modell oder die nächster Nachbar Suche, anstelle der geschätzten Attributvektoren oder anstelle der PHOC des Lexikons, zu trainieren. Dies könnte Overfitting vermeiden. Ferner scheinen die geringeren Regularisierungsparameter und Dimensionalitäten nicht förderlich für die CCA. In der Regel führen erst Regularisierungsparameter größer 10000 zu vergleichsweise guten Ergebnissen im Rahmen der Kreuzvalidierung und geringe Dimensionalitäten können in einem hohen Informationsverlust oder starkem Overfitting auf den Trainingsdaten resultieren. Das in dieser Arbeit vorgestellte Klassifikationsverfahren über CCA resultierte in den schlechtesten Ergebnissen im Rahmen der Evaluierung der Klassifikatoren. Angesichts der gewonnen Erkenntnisse zum Verfahren, ist dies wahrscheinlich dadurch bedingt, dass die Parameter der CCA unzureichend gewählt wurden. Es kann die Erkenntnis gewonnen werden, dass es hinsichtlich der Kreuzvalidierung, sowie der Wahl von Trainingsdaten für die CCA einer Verbesserung Bedarf, damit ein solcher Klassifikationsansatz mit Ausgaben des PHOCNet gute Ergebnisse erzielt.

LITERATURVERZEICHNIS

[ABC+06] Augustin, Emmanuel ; Brodin, Jean-marie ; Carré, Matthieu ; Geoffrois, Edouard ; Grosicki, Emmanuèle ; Prêteux, Françoise: RIMES evaluation campaign for handwritten mail processing. In: *Workshop on Frontiers in Handwriting Recognition*, 2006

[AGFV14] Almazán, Jon ; Gordo, Albert ; Fornés, Alicia ; Valveny, Ernest: Word Spotting and Recognition with Embedded Attributes. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36 (2014), Nr. 12, S. 2552–2566

[Bis06] Bishop, Christopher M.: *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg : Springer-Verlag, 2006

[BNK13] Bluche, T. ; Ney, H. ; Kermorvant, C.: Tandem HMM with convolutional neural network for handwritten word recognition. In: *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, S. 2390–2394

[DHS11] Duchi, John C. ; Hazan, Elad ; Singer, Yoram: Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. In: *Journal of Machine Learning Research* 12 (2011), S. 2121–2159

[DKMJ18] Dutta, K. ; Krishnan, P. ; Mathew, M. ; Jawahar, C. V.: Improving CNN-RNN Hybrid Networks for Handwriting Recognition. In: *International Conference on Frontiers in Handwriting Recognition (ICFHR)*, 2018, S. 80–85

[DKN14] Doetsch, Patrick ; Kozielski, Michal ; Ney, Hermann: Fast and Robust Training of Recurrent Neural Networks for Offline Handwriting Recognition. In: *International Conference on Frontiers in Handwriting Recognition (ICFHR)* 2014 (2014), December, S. 279–284

[FEHF09] Farhadi, Ali ; Endres, Ian ; Hoiem, Derek W ; Forsyth, David Alexander: Describing objects by their attributes. In: *IEEE Computer Society*

Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, January 2009, S. 1778–1785

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

[GFGS06] Graves, Alex ; Fernández, Santiago ; Gomez, Faustino ; Schmidhuber, Jürgen: Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In: *International Conference on Machine Learning (ICML)*. New York, NY, USA : ACM, 2006, S. 369–376

[GSGN17] Giotis, Angelos ; Sfikas, Giorgos ; Gatos, Basilis ; Nikou, Christophoros: A Survey of Document Image Word Spotting Techniques. In: *Pattern Recognition* (2017), February

[GWFM+13] Goodfellow, Ian J. ; Warde-Farley, David ; Mirza, Mehdi ; Courville, Aaron ; Bengio, Yoshua: Maxout Networks. In: *International Conference on Machine Learning (ICML)* Bd. 28, 2013, S. III–1319–III–1327

[Hoc98] Hochreiter, Sepp: The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6 (1998), Nr. 2, S. 107–116

[Hot35] Hotelling, H.: The Most Predictable Criterion. In: *Journal of Educational Psychology* 26 (1935), S. 139 – 152

[Hot36] Hotelling, H.: Relations Between Two Sets of Variates. In: *Biometrika* 28 (1936), S. 321 – 377

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

[HZRS15a] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In: *International Conference on Computer Vision (ICCV)* (2015), S. 1026 – 1034

[HZRS15b] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Spatial pyramid pooling in deep convolutional networks for visual recognition.

In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37 (2015), Nr. 9, S. 1904 – 1916

[KB14] KINGMA, Diederik P. ; BA, Jimmy: Adam: A Method for Stochastic Optimization. In: *International Conference on Learning Representations (ICLR)* (2014)

[KJ16] KRISHNAN, Praveen ; JAWAHAR, C. V.: Matching Handwritten Document Images. In: *European Conference on Computer Vision (ECCV)*, 2016, S. 766–782

[Knu92] KNUTH, Donald E.: Two Notes on Notation. In: *American Mathematical Monthly* 99 (1992), Mai, Nr. 5, S. 403–422

[LJY19] LI, Fei-Fei ; JOHNSON, Justin ; YEUNG, Serena: *Stanford Lecture Notes to CS231n: Convolutional Neural Networks for Visual Recognition*. http://cs231n.stanford.edu. Version: 2019

[LNH09] LAMPERT, CH. ; NICKISCH, H. ; HARMELING, S.: Learning To Detect Unseen Object Classes by Between-Class Attribute Transfer. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. Piscataway, NJ, USA : IEEE Service Center, June 2009, S. 951–958

[LNH14] LAMPERT, Christoph H. ; NICKISCH, Hannes ; HARMELING, Stefan: Attribute-Based Classification for Zero-Shot Visual Object Categorization. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36 (2014), March, Nr. 3, S. 453–465

[LZD+08] LI, Yi ; ZHENG, Yefeng ; DOERMANN, David ; JAEGER, Stefan ; LI, Yi: Script-Independent Text Line Segmentation in Freestyle Handwritten Documents. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30 (2008), August, Nr. 8, S. 1313–1329

[MB02] MARTI, Urs-Viktor ; BUNKE, H: The IAM-database: An English sentence database for offline handwriting recognition. In: *International Journal on Document Analysis and Recognition (IJDAR)* 5 (2002), November, S. 39–46

[MHN13] MAAS, Andrew L. ; HANNUN, Awni Y. ; NG, Andrew Y.: Rectifier nonlinearities improve neural network acoustic models. In: *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013

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

[Nie18] Nielsen, Michael A.: *Neural Networks and Deep Learning*. Version: 2018. http://neuralnetworksanddeeplearning.com/

[PF09] Plötz, Thomas ; Fink, Gernot A.: Markov Models for Offline Handwriting Recognition: A Survey. In: *International Journal on Document Analysis and Recognition (IJDAR)* 12 (2009), November, Nr. 4, S. 269–298

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

[RM07] Rath, Tony M. ; Manmatha, R.: Word spotting for historical documents. In: *International Journal of Document Analysis and Recognition (IJDAR)* 9 (2007), April, Nr. 2, S. 139–152

[Ros58] Rosenblatt, F.: The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. In: *Psychological Review* (1958), S. 65–386

[RRMF18] Rusakov, Eugen ; Rothacker, Leonard ; Mo, Hyunho ; Fink, Gernot A.: A Probabilistic Retrieval Model for Word Spotting Based on Direct Attribute Prediction. In: *International Conference on Frontiers in Handwriting Recognition (ICFHR)*, 2018, S. 38–43

[RSS+18] Retsinas, George ; Sfikas, Giorgos ; Stamatopoulos, Nikolaos ; Louloudis, Georgios ; Gatos, Basilis: Exploring Critical Aspects of CNN-based Keyword Spotting. A PHOCNet Study. In: *International Workshop on Document Analysis Systems (DAS)*, 2018, S. 13–18

[SF18] Sudholt, Sebastian ; Fink, Gernot A.: Attribute CNNs for Word Spotting in Handwritten Documents. In: *International Journal on Document Analysis and Recognition (IJDAR)* 21 (2018), September, Nr. 3, S. 199–218

[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), S. 1929–1958

[SMDH13] SUTSKEVER, Ilya ; MARTENS, James ; DAHL, George ; HINTON, Geoffrey: On the Importance of Initialization and Momentum in Deep Learning. In: *International Conference on Machine Learning (ICML)* Bd. 28, 2013, S. III–1139–III–1147

[SRG16] SFIKAS, Giorgos ; RETSINAS, George ; GATOS, Basilis: Zoning Aggregated Hypercolumns for Keyword Spotting. In: *International Conference on Frontiers in Handwriting Recognition (ICFHR)*, IEEE Computer Society, 2016, S. 283 – 288

[Vin76] VINOD, H. D.: Canonical ridge and econometrics of joint production. In: *Journal of Econometrics* 4 (1976), May, Nr. 2, S. 147 – 166