technische universität
dortmund

Domain Adaptation für das Word Spotting mit Attribute CNNs

Bachelorarbeit

Benjamin Kraatz
22. Oktober 2020

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

Fakultät für Informatik
Technische Universität Dortmund
https://www.cs.tu-dortmund.de

INHALTSVERZEICHNIS

1 EINLEITUNG

Die digitale Erschließung von Wissen, das in Form von handschriftlich verfassten Dokumenten vorliegt, stellt nach wie vor eine große Herausforderung dar. Unmengen von Sammlungen handschriftlicher Dokumente warten darauf, nicht nur digitalisiert, sondern auch indexiert und folglich für Suchmaschinen auffindbar gemacht zu werden. Trotz der erstaunlichen, über Jahrtausende entwickelten Fähigkeit des Menschen, Muster mit annähernder Perfektion zu erkennen und zu deuten, verlangt die genannte Aufgabe nach automatischen Verfahren, da der Mensch die erkannten Informationen nicht effizient genug weiterverarbeiten kann. Maschinelle Verfahren verfügen nicht annähernd über die Fähigkeit des Menschen, Muster zu erkennen und mit einer Bedeutung zu verknüpfen, jedoch gleicht sich dieser Nachteil durch Schnelligkeit in der Verarbeitung immenser Informationsmengen aus. So wurden im Laufe der Jahre zum einen automatische Verfahren entwickelt, die Text erkennen und so bei der Digitalisierung helfen, und zum anderen Verfahren, die Dokumente anhand von Schlüssenbegriffen indexieren.

Letzteres Konzept wird auch als Word Spotting bezeichnet und nimmt in dieser Arbeit einen zentralen Platz ein. Die Grundidee lautet dabei, Anfragewörter in einer Dokumentensammlung aufzufinden. Damit dies digital möglich ist, werden Dokumente zunächst eingescannt. Die so erzeugten digitalen Abbilder der Dokumente können dann beispielsweise segmentierungsbasiert in Wortabbilder unterteilt werden. Diese Wortabbilder bilden jeweils einen Teil des Dokuments ab, auf dem ein Wort steht. Die resultierende Sammlung von Wortabbildern dient als Basis für Anfragen, die schließlich automatisch mit allen vorhandenen Wortabbildern verglichen werden, um möglichst ähnliche Kandidaten als Antwort auf Anfragen liefern zu können.

Eines dieser automatischen Verfahren geht auf die rasanten Fortschritte im maschinellen Sehen zurück, das stark durch die Verpflechtung mit Maschine Learning profitieren konnte. Sogenannte Faltungsnetze (Convolutional Neural Networks - CNNs), eine spezielle Form künstlicher neuronaler Netze, können anhand von annotierten Trainingsdaten die Klassifikation neuer, trainingsfremder Daten lernen. Sie eignen sich besonders gut zur Analyse von Bildern [GBC16]. Faltungsnetze konnten in den letzten Jahren ihre Leistungsfähigkeit insbesondere im Bereich der Document Image Analysis demonstrieren [SF18]. Jedoch ist es nach wie vor teuer und aufwändig, manuell annotierte Trainingsdaten zu erstellen, was die große Schwachstelle dieses Ansatzes offenbart.

Im Mittelpunkt dieser Arbeit steht die Nutzung synthetischer Daten als Alternative zu realen Daten für das Training von Faltungsnetzen für Word Spotting. Synthetische

Trainingsdaten sollen helfen, das Problem des kostspieligen Annotationsaufwandes realer Trainingsdaten zu umgehen. Sie sind zwar vergleichsweise einfach zu erstellen, dennoch ist die Nutzung synthetischer Trainingsdaten nicht unproblematisch, da sich die Merkmale, die ein CNN aus ihnen extrahieren kann, deutlich von extrahierten Merkmalen realer Daten unterscheiden. Anhand dreier CNN Standardarchitekturen wird daher untersucht, wie hoch die Leistungsverluste bei einem Domänenwechsel von synthetischen Trainingsdaten auf reale Testdaten sind. Des Weiteren wird ein CNN vorgestellt, das imstande ist, Domain Adaptation auszuführen. Unter Domain Adaptation wird die Anpassung der Trainingsdatendomäne an eine neue Domäne verstanden [Kan+20][GL15]. Dieses neue CNN, das DAPHOCNet, basiert auf dem sogenannten PHOCNet. Dabei handelt es sich um ein für Word Spotting entwickeltes CNN, das Wortabbilder auf Attributrepräsentationen abbildet [SF18]. In verschiedenen Experimenten wird überprüft, inwieweit sich die Leistungsfähigkeit eines CNNs im Kontext von synthetischen Trainingsdaten durch die Anpassung an eine neue Domäne verändert.

Diese Arbeit ist wie folgt gegliedert: Kapitel 2 führt in die Grundlagen ein, die für die Methodik dieser Arbeit notwendig sind. Zu Beginn werden die zentralen Konzepte und Ansätze von Word Spotting erläutert, da dieser Forschungsbereich des maschinellen Sehens im Mittelpunkt dieser Arbeit steht. Darauf folgt eine Einführung in die grundlegenden Bestandteile und Funktionsweisen künstlicher neuronaler Netze sowie des Spezialfalls der Faltungsnetze, die in der Methodik dieser Arbeit verwendet und modifiziert werden. Die Grundlagen werden mit der Diskussion verschiedener Lernverfahren abgeschlossen, die im Zuge des Trainings künstlicher neuronaler Netze Anwendung finden.
Kapitel 3 stellt eng verwandte Arbeiten vor. Zum einen wird das speziell für Word Spotting entwickelte PHOCNet präsentiert und zum anderen das Konzept der Domain Adaptation.
In Kapitel 4 wird die Methodik dieser Arbeit diskutiert, die im Anschluss in Kapitel 5 experimentell ausgewertet und mit der Literatur verglichen wird.
Zum Schluss wird in Kapitel 6 ein Fazit zu den Erkenntnissen aus dieser Arbeit gezogen und diskutiert, welche zukünftigen Arbeiten in diesem Kontext sinnvoll erscheinen.

2 GRUNDLAGEN

2.1 WORD SPOTTING

Word Spotting (auch: Keyword Spotting) ist eine Methode, die zur Indexierung von Dokumenten genutzt wird, die in Form von digitalen Bildern vorliegen und für die andere Verfahren, insbesondere Optical Character Recognition (OCR), keine zufriedenstellenden Ergebnisse liefern. OCR soll eingescannte Dokumente zu ASCII-Text konvertieren, der anschließend mittels Text Retrieval Systemen indexiert werden kann [MHR96]. Dieses Verfahren ist aber auf historischen und handgeschriebenen Dokumenten ineffizient, da altersbedingte Schädigungen des Papiers (die sich auch auf das digitale Abbild übertragen) und unbekannte Fonts sowie Handschriftvariationen vorkommen [Gio+17].

Die schiere Menge an verfügbaren, aber noch nicht indexierten historischen und handschriftlichen Dokumentenabbildern sorgen für ein stetig wachsendes Interesse an erkennungsfreien Image Retrieval Verfahren, die unter dem Begriff Word Spotting zusammengefasst werden [Gio+17]. Das Ziel dieser Verfahren ist es, Instanzen von Wortabbildern aus einer Menge von eingescannten Dokumenten zurückzuliefern, die bezüglich einer Nutzeranfrage (Query) am relevantesten sind. Rückgaben von Word Spotting Verfahren, die Instanzen von hoher Relevanz bezüglich einer Anfrage enthalten sollen, lassen sich einfach interpretieren, was einen weiteren Punkt dafür liefert, Word Spotting auf handgeschriebenen Dokumenten OCR vorzuziehen. Die Güte einer Rückgabeliste (Retrieval List) eines Word Spotting Verfahrens wird mit einer Methode namens mean Average Precision (mAP) gemessen, die in Kapitel 5.2 erläutert wird.

Anfragen werden in zwei Kategorien eingeteilt, Query by Example (QbE) und Query by String (QbS). QbE bezeichnet Anfragen in Form von Wortabbildern. Sind Anfragen hingegen Strings, so wird von QbS gesprochen [Gio+17][AAKM16]. QbS gewinnt in der Praxis immer mehr Relevanz, da für QbE erst ein Beispielwortabbild in einer Dokumentensammlung gefunden werden muss, wodurch das Problem eventuell schon gelöst wird [SF18]. Neben segmentierungsbasierten Word Spotting Verfahren, bekannt durch die Arbeit von *Manmatha et al.* [MHR96], gibt es segmentierungsfreie Verfahren, die nach und nach ein Fenster über das zu bearbeitende Dokument schieben, um Wortabbilder zu gewinnen (siehe [AAKM16]).

Word Spotting stammt aus dem Bereich der Spracherkennung und wurde auf die Domäne historischer sowie handschriftlicher Dokumente übertragen. Den ersten An-

satz dazu präsentierten *Manmatha et al.* [MHR96] im Jahr 1995. Zuerst werden die Seiten eines Dokuments segmentiert, das heißt, dass jedes Segment ein Wortabbild enthält. Im Anschluss werden die Wortabbilder mittels paarweisem Matching in Äquivalenzklassen eingeteilt [MHR96]. Danach muss der Nutzer entsprechende ASCII Gegenstücke für die Repräsentanten ausgewählter Äquivalenzklassen bereitstellen, wodurch Wörter in diesen Klassen schließlich indexiert werden können.
Die größte Schwierigkeit dieses Ansatzes birgt das Matching der Wortabbilder, da Handschriftvariation schwer zu modellieren ist [MHR96]. Matching bezeichnet dabei die paarweise Berechnung der Ähnlichkeit von Wortabbildern. Paare, die auf Ähnlichkeit untersucht werden, bestehen aus einer Anfrage und einem Wortabbild aus dem segmentierten Dokument. Grundsätzlich werden die Wortabbilder vor dem Matching mittels Merkmalsextraktion auf Merkmalsvektoren abbildet und somit in andere Repräsentationen überführt, die im Anschluss statt der rohen Wortabbilder zur Ähnlichkeitsberechnung verwendet werden. Die Berechnung der Ähnlichkeit zweier Merkmalsvektoren wird auf Basis eines Distanzmaßes wie beispielsweise der Kosinusähnlichkeit (Cosine Similarity) durchgeführt. Die Repräsentation eines Wortabbildes durch extrahierte Merkmale benötigt weniger Speicherplatz und sorgt für eine höhere Leistungsfähigkeit des Systems [AAKM16].

Auf den ersten Ansatz von *Manmatha et al.* folgten unter anderem sequentielle Methoden, die erfolgreich in der Handschrifterkennung angewendet und auf Word Spotting übertragen wurden. Dazu gehören beispielsweise Hidden Markov Models (HMM) [RS09], Dynamic Time Warping [RM07] und Long Short Term Memory Networks (LSTM) [Fri+12].

Neben sequentiellen Modellen finden Ansätze, die lokale Deskriptoren verwenden, häufig Anwendung. Zu lokalen Deskriptoren zählen beispielsweise HOG-basierte Deskriptoren [AFV13] sowie SIFT-Deskriptoren [Low04], die vorallem für Image Retrieval Verfahren, die auf dem erfolgreichen Bag of Features Modell [AD07][Rus+15][Rus+11] basieren, verwendet werden. Wortabbilder werden dabei durch eine Menge von Visual Terms (Bag of Visual Words) repräsentiert [AD07].

Ein weiteres Konzept zur Repräsentation von Eingaben, die Attributrepräsentation, wurde von *Almazán et al.* [Alm+14] vorgestellt. Aus Wortabbildern extrahierte Merkmale (QbE) und Texteingaben (QbS) werden in einen gemeinsamen Attributraum projiziert, der die Grundlage für Ähnlichkeitsberechnungen mittels Kosinusdistanz bezüglich einer Anfrage und den zu überprüfenden Wortabbildern eines Datensatzes bildet [SF18]. Diese Einbettung wird als PHOC bezeichnet und steht für Pyramidal Histogram of Characters (siehe dazu Kapitel 3.1).

Große Leistungsfortschritte wurden dank erstmaliger Anwendung von Convolutional Neural Networks zur Erkennung (Recognition) von handschriftlichen Ziffern durch *LeCun et al.* [LeC+89] erzielt. Es folgten weitere erfolgreiche Anwendungen von CNNs im Recognition-Bereich [KSH12], die *Sudholt et al.* zu ihrem TPP-PHOCNet [SF17] inspirierten, das wiederum ein Grundbaustein der in dieser Arbeit präsentierten

Methodik (siehe Kapitel 4) ist. Das TPP-PHOCNet ist ein CNN, das auf der PHOC-Attributrepräsentation von *Almazán et al.* [Alm+14] basiert. Es lernt, Wortabbilder auf PHOC-Attributrepräsentationen abzubilden [SF17].

2.2 NEURAL NETWORKS

Künstliche neuronale Netze (KNNs) sind Berechnungsmodelle zur Funktionsapproximation, die dem Gebiet des maschinellen Lernens zuzuordnen sind. Als grobes Vorbild dient das Nervensystem, ein biologisches neuronales Netz [Yeg04]. Die Grundbausteine neuronaler Netze sind Neuronen. In der Biologie wird auch von Nervenzellen gesprochen, die sich durch die Fähigkeit der Erregungsübertragung auszeichnen. Der Anatom *Heinrich von Waldeyer-Hartz* prägte dabei den Begriff Neuron [Meh+19]. Entscheidend für die Übertragung des Begriffs auf Berechnungsmodelle ist aber einzig die Bedeutung und insbesondere das Konzept einer Nervenzelle als Informationsverarbeitungseinheit.

McCulloch und Pitts veröffentlichten in ihrer Arbeit aus dem Jahr 1943 als erste ein künstliches Neuron als logische Schwellwerteinheit [MP43]. Das sogenannte McCulloch-Pitts-Neuron ist in der Lage, n binäre Signale als Eingabe zu bekommen und ein einzelnes binäres Signal auszugeben. Die Ausgabe hängt davon ab, ob die aufsummierten Eingaben einen reellwertigen Schwellwert θ erreichen. Mit McCulloch-Pitts-Neuronen lassen sich einfache boolsche Funktionen wie AND-, OR- und NOT modellieren [MP43].

Im Jahr 1958 stellte *Rosenblatt* das Perzeptron [Ros58] vor. Dabei handelt es sich um ein KNN, das in seiner Ursprungsform nur aus einem einzigen Neuron besteht, das auf dem McCulloch-Pitts-Neuron basiert (siehe Abbildung 2.2.1).

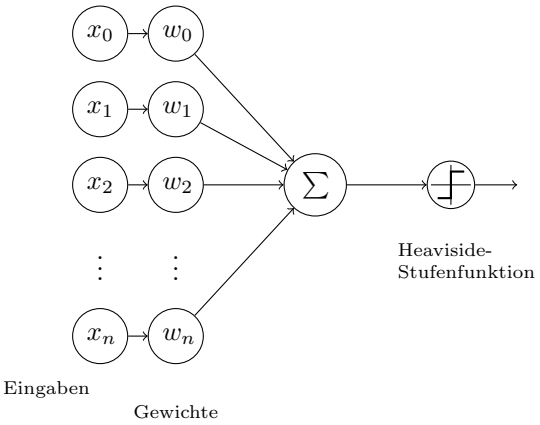


Abbildung 2.2.1: Einfaches Perzeptron: Übersteigt die gewichtete Summe der Komponenten des Eingabevektors $\mathbf{x}$ und des Bias b den Schwellwert θ (siehe Formel 2.2.1), findet eine Aktivierung statt und das Neuron gibt den Wert 1 aus.

$$f(\mathbf{x}) = \begin{cases} 1 & \text{falls } \mathbf{w}^\top \mathbf{x} + b > \theta \\ 0 & \text{sonst} \end{cases} \tag{2.2.1}$$

Typischerweise wird ein einfaches Perzeptron zu binärer linearer Klassifizierung genutzt und es stößt an seine Grenzen, wenn ein Problem nicht linear separierbar ist. *Minksy und Papert* zeigten 1969, dass das einfache Perzeptron nach *Rosenblatt* die XOR-Funktionen nicht berechnen kann [MP69]. Es war jedoch zu der Zeit schon klar, dass mit größeren KNN auch nichtlineare Funktionen berechnet werden können. *Minksy und Papert* nannten in diesem Kontext die Möglichkeit zum Bau einer universellen Turingmaschine durch Kombination vieler McCulloch-Pitts-Neuronen, die einzeln nur elementare boolsche Funktionen implementieren können [MP69]. Generell zeigt sich die Mächtigkeit neuronaler Netze vorallem dann, wenn sie aus einer Vielzahl von Neuronen bestehen.

Sogenannte tiefe neuronale Netze haben im Laufe der Zeit in immer mehr Anwendungsbereichen, insbesondere im maschinellen Sehen, an Bedeutung gewonnen. Der Begriff Tiefe ist im Kontext künstlicher neuronaler Netze nicht eindeutig definiert, wird aber häufig über die Anzahl der Neuronenschichten (Layers) beschrieben [GBC16]. Die Neuronen einer Schicht sind dabei in der Regel mit Neuronen der Nachfolgeschicht verbunden. Bereits zwei Schichten reichen aus, um beliebige Funktionen zu approximieren, aber der Trend zeigt in Richtung tieferer Netze [GBC16].

2.2.1 *Multi Layer Perceptron*

Ein Multilayer Perceptron (MLP) ist ein neuronales Netzwerk, das zusätzlich zur Ausgabeschicht mindestens eine verdeckte Schicht enthält.
MLPs beinhalten zwar den von *Rosenblatt* geprägten Begriff Perzeptron [Ros58], jedoch handelt es sich im modernen Sinne dabei nicht um eine Vernetzung einfacher Perzeptren nach *Rosenblatt*, da heute statt der Heaviside-Stufenfunktion andere Aktivierungsfunktionen bevorzugt werden, von denen für diese Arbeit relevante Varianten in Kapitel 2.2.2 diskutiert werden.

MLPs werden auch als Feedforward Networks bezeichnet [GBC16]. Feedforward Netzwerke verdanken ihren Namen der Art, wie Informationen durch sie fließen. Die Ausgaben der Neuronen einer Schicht werden nur an Neuronen, die in Nachfolgeschichten liegen, weitergeleitet. Handelt es sich um die Ausgabe der letzten Schicht, so ist dies die Ausgabe des Netzwerks. Es gibt in Feedforward Netzwerken keine Feedback Verbindungen, das heißt Ausgaben fließen nicht wieder in ihre ursprüngliche Schicht oder davorliegende Schichten mit ein [GBC16].

Die Informationen, die in das Netzwerk eingegeben werden, durchlaufen Sequenzen von Neuronen, wobei jedes Neuron eine Funktion darstellt. Insgesamt ist jedes Feedforward Netzwerk daher eine Abbildung $f(x) = y$, bei der x für den Eingabevektor und y für die Ausgabe des Netzwerks steht. Die Abbildung (auch: Mapping) $f(x)$ hat die Eigenschaft, dass sie sich aus der Komposition der verschiedenen Funktionen entlang der Schichten ergibt [GBC16]. Abbildung 2.2.2 zeigt ein Beispiel für ein MLP.

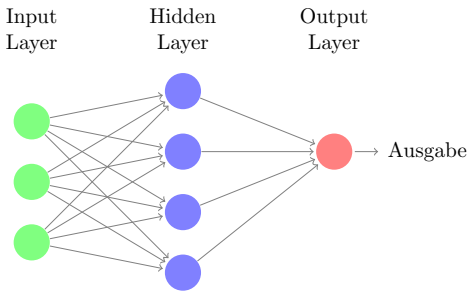


Abbildung 2.2.2: Einfaches Multi Layer Perceptron mit drei Eingabeneuronen, einer verborgenen Schicht mit vier Neuronen und einer einzelnen Ausgabe.

2.2.2 *Aktivierungsfunktionen*

Aktivierungsfunktionen transformieren Eingabesignale von Neuronen zu Ausgabesignalen [Sha17]. Bei Eingaben für Aktivierungsfunktionen handelt es sich jeweils um die Summe der Produkte der Ausgaben der vorherigen Schichten und den zugehörigen Gewichten. Aktivierungsfunktionen definieren daher, ob und falls ja, wie stark ein Neuron aktiviert wird und somit auch, wie die Eingabe der nächsten verbundenen Schicht aussieht. Die Heaviside-Stufenfunktion (Abb. 2.2.3) stellt die einfachste aller Aktivierungsfunktionen dar, weil durch ihre in Gleichung 2.2.2 definierte Abstufung zwischen 0 und 1 nur eine vollständige Aktivierung oder eben gar keine Aktivierung stattfinden kann.

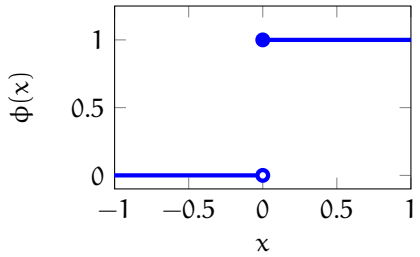
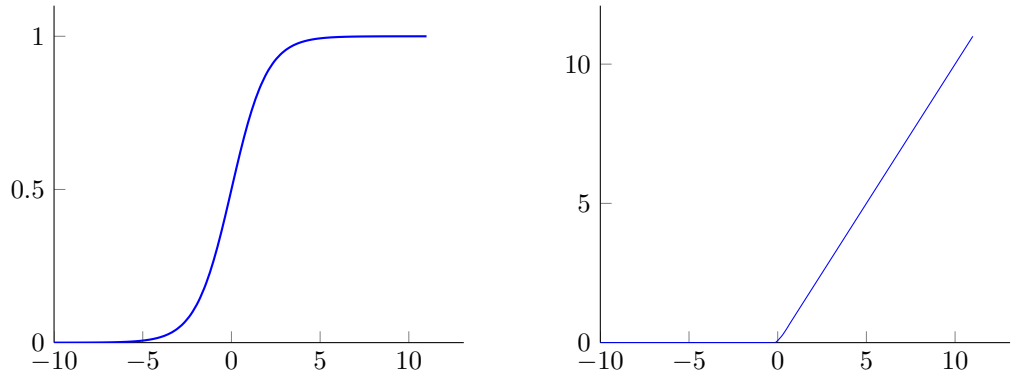


Abbildung 2.2.3: Heaviside-Stufenfunktion

$$\phi(x) = \begin{cases} 0 & \text{falls } x < 0 \\ 1 & \text{falls } x \geqslant 0 \end{cases} \tag{2.2.2}$$

Zur Approximation nichtlinearer Funktionen ist es jedoch notwendig, nichtlineare Aktivierungsfunktionen (auch: Non-linearities) zu nutzen, da KNNs sonst nur in der Lage sind, Eingaben linear auf Ausgaben abzubilden, wodurch es unmöglich wird, nicht-triviale Eigenschaften der Daten zu extrahieren [Sha17].
Im Folgenden werden zwei nichtlineare Aktivierungsfunktionen vorgestellt, die für diese Arbeit relevant sind. Für Informationen über andere Aktivierungsfunktionen siehe [KO11][RZL17].



(a) Sigmoid-Aktivierungsfunktion (b) ReLU-Aktivierungsfunktion

Abbildung 2.2.4: Darstellung der für diese Arbeit relevanten Aktivierungsfunktionen

Die Sigmoid-Aktivierungsfunktion (Abb. 2.2.4a) ist eine differenzierbare Approximation der Heaviside-Stufenfunktion. Sie wird im PHOCNet (Kapitel 3.1) und DAPHOCNet (Kapitel 4.2) jeweils in der letzten Schicht des MLPs eingesetzt, das Attributvektoren approximieren soll. Der Grund für die Bevorzugung der Sigmoid-Aktivierungsfunktion gegenüber anderen, häufig verwendeten Aktivierungsfunktionen, wie beispielsweise der Softmax-Aktivierungsfunktion, ist die Aufgabe, PHOC-Attributvektoren zu approximieren, was eine Multi-Label Klassifikation darstellt. Multi-Label Klassifikation ist mit der Softmax-Aktivierungsfunktion nicht möglich, wie in Kapitel 3.1 erläutert wird.

Die Sigmoid-Aktivierungsfunktion geht auf die logistische Funktion aus dem 19. Jahrhundert zurück, die als Modell für Populationswachstum eingeführt wurde [Cra02]. Der S-förmige Kurvenverlauf der in Gleichung 2.2.3 definierten Funktion sorgt für eine weiche Aktivierung. Es gibt daher im Gegensatz zur Heaviside-Stufenfunktion kein All-or-nothing Verhalten.

$$\mathrm{sigmoid}(\mathbf{x}) = \frac{1}{1 + e^{-x}} \tag{2.2.3}$$

Die Ausgabe der Sigmoid-Aktivierungsfunktion kann als Pseudowahrscheinlichkeit für eine Klassenzugehörigkeit interpretiert werden. Vor dem Durchbruch der ReLU-Aktivierungsfunktion, die im nachfolgenden Abschnitt diskutiert wird, wurde die Sigmoid-Aktivierungsfunktion aufgrund einfacher Diffenrenzierbarkeit verwendet. Diffenrenzierbarkeit ist eine wichtige Eigenschaft von Aktivierungsfunktionen im Kontext der Optimierung (siehe Kapitel 2.2.4) künstlicher neuronaler Netzwerke. Die Nutzung der Sigmoid-Aktivierungsfunktion in allen Schichten tiefer neuronaler Netze ist im Gegensatz zur Nutzung der ReLU-Aktivierungsfunktion aufgrund von

Gradienten, die verschwindend gering werden, problematisch. Gewichte erhalten dadurch keine signifikanten Updates mehr. Dieses Problem nennt sich Vanishing Gradient [Hoc98] und kann bei Gradienten-basierten Optimierungsverfahren (siehe Kapitel 2.2.4) auftreten.

Die ReLU-Aktivierungsfunktion (Rectified Linear Unit - 2.2.4b) ist gegenwärtig die meistverwendete und erfolgreichste Aktivierungsfunktion [RZL17]. Neuronen, die diese Aktivierungsfunktion nutzen, werden auch einfach als Rectified Linear Unit bezeichnet [GBC16].

$$\mathrm{ReLU}(\mathbf{x}) = \max(0, x) \tag{2.2.4}$$

Gleichung 2.2.4 definiert, dass negative Eingaben auf 0 abgebildet werden und sich die ReLU-Funktion bei positiven Eingaben wie die Identitätsfunktion verhält. Sie ist stückweise linear definiert, aber insgesamt nichtlinear. KNNs, die ReLUs enthalten, sind leichter zu optimieren als Netze, die die Sigmoid-Aktivierungsfunktion nutzen [RZL17], da die für ReLU benötigten Berechnungen einfacher sind, was insgesamt zu schnellerem Training führt. Insbesondere löst ReLU das Vanishing Gradient Problem. Bei Verwendung der ReLU-Aktivierungsfunktion werden Gradienten mit wachsender Tiefe des KNN nicht verschwindend gering.

Im Jahr 2011 wurde gezeigt, dass die Verwendung von ReLUs in KNNs mit mehr als drei Hidden Layers die Möglichkeit eröffnet, solche KNNs von Grund auf überwacht (supervised - siehe Kapitel 2.4) zu trainieren, das bedeutet komplett ohne unüberwachtes (unsupervised - siehe Kapitel 2.4) Pre-Training [GBB11]. Diese Erkenntnis ermöglichte das vollständig überwachte Training tiefer neuronaler Netze, die als state-of-the-art gelten [KSH12]. Die ReLU-Aktivierungsfunktion wird in allen verdeckten Schichten des DAPHOCNets (siehe Kapitel 4.2) eingesetzt.

2.2.3 *Fehlerfunktionen*

Der Lernprozess künstlicher neuronaler Netze basiert auf dem Prinzip, aus Fehlern zu lernen, um bestimmte Zielwerte (Targets) nach und nach besser zu approximieren, das heißt, ausgehend von einem Fehler(-wert), Änderungen an den Parametern des Netzes vorzunehmen, um den Fehler zu minimieren. Zur Berechnung eines Fehlers wird eine Funktion genutzt, die die Differenz von Approximation und Zielwert, welcher der gewünschten Ausgabe entspricht, abbildet. Allgemein wird dieses Maß als Criterion oder Objective Function bezeichnet [GBC16]. Lautet das Ziel, diese Funktion zu minimieren, wird stattdessen eine der Bezeichnungen Cost Function, Loss

Function oder Error Function verwendet [GBC16]. Als einfaches Beispiel eignet sich die Mean-Squared Error-Funktion (MSE).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y} - y)_i^2 \tag{2.2.5}$$

Gleichung 2.2.5 gibt die mittlere quadrierte Differenz aus Approximation $\hat{y}$ und Target y an. Seien beispielsweise Target y und Approximation $\hat{y}$ in Form binärer One-Hot Kodierungen angegeben. Mit der One-Hot Kodierung lässt sich die Zugehörigkeit zu einer Klasse als Vektor kodieren. One-Hot bedeutet, dass genau ein Element des Vektors den Wert 1 hat und die restlichen Elemente auf 0 gesetzt sind. Es wird genau eine Klasse zugewiesen. Im Fall der MSE-Funktion würde nun für Approximation-Target-Paare die Summe quadrierter Differenzen berechnet werden, die durch die Anzahl der Paare geteilt wird, um einen Fehlerwert zu erhalten.

Sind statt genau einem Label mehrere Label pro Datenpunkt möglich, lautet das Ziel Multi-Label Klassifikation. Das ist bei der Approximation von PHOCs der Fall (siehe Kapitel 3.1). Dafür eignet sich die Binary Cross-Entropy Funktion (BCE), da sie ideal für Variablen mit on-off-Charakteristik ist [CAB17].

$$BCE = -\frac{1}{n}\sum_{i=1}^{n} y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i) \tag{2.2.6}$$

Gleichung 2.2.6 definiert die BCE-Funktion, wobei $\hat{y}_i$ für den i-ten Ausgabewert steht und y_i für den korrespondierenden Zielwert. Für alle Ausgaben gibt es nur zwei mögliche Klassen. Im Kontext von Word Spotting mit Attribute CNNs und der damit verbundenen Multi-Label Klassifikation bedeutet das, dass das i-te Elemente eines Attributvektors (PHOC) in der Berechnung der BCE-Funktion als y_i bzw. $\hat{y}_i$ bezeichnet wird. Hat das i-te Element den Wert 1, gehört es zur i-ten Klasse. Ist der Wert hingegen 0, gehört das i-te Element nicht zur i-ten Klasse.

2.2.4 *Optimierung*

Künstliche neuronale Netze zu trainieren bedeutet, Optimierungsprobleme zu lösen [GBC16]. Konkret bedeutet das, die Parameter θ eines KNNs so zu verändern, dass im Gegenzug die verwendete Kostenfunktion $J(\theta)$ minimiert wird [GBC16].

Ein weit verbreiteter Optimierungsalgorithmus ist das Gradientenabstiegsverfahren (Gradient Descent) [GBC16]. Ziel des Gradientenabstiegsverfahrens, die Minimierung einer Kostenfunktion, wird durch iterative Änderungen der Parameter θ eines KNNs

erreicht. Ein solches Update der Parameter wird auch als Optimierungsschritt bezeichnet. Die Berechnung des Gradienten der Kostenfunktion $\nabla_\theta J$ ist dabei von elementarer Bedeutung, da der Gradient die Richtung eines Updates angibt. Ein Update der Parameter θ (Formel 2.2.7) ergibt sich durch Subtraktion des mit einer Lernrate η multiplizierten Gradienten von θ. Die Lernrate η ist ein Faktor, mit dessen Hilfe sich die Größe des Updates steuern lässt. Für gewöhnlich handelt es sich dabei um einen kleinen Faktor, was für kleine Änderungen an den Gewichten sorgt.

$$\theta' = \theta - \eta \nabla_\theta J \tag{2.2.7}$$

Alternativ zum Gradientenabstiegsverfahren kann Stochastic Gradient Descent (SGD) zur Optimierung verwendet werden. Der Unterschied liegt in der Häufigkeit der Optimierungsschritte. Im Allgemeinen wird bei Gradient Descent erst dann der Optimierungsschritt durchgeführt, wenn alle Trainingsbeispiele durch das Netz propagiert wurden und das Mittel der Gradienten für die einzelnen Trainingsbeispiele berechnet wurde. SGD hingegen führt nach jedem Trainingsbeispiel einen Optimierungsschritt aus, was rechenintensiver ist und somit für eine längere Trainingsdauer sorgt [GBC16][Nie18]. Ein häufig verwendeter Kompromiss aus beiden Varianten lautet, Minibatches aus der Gesamtmenge der Trainingsbeispiele zu bilden und nach dem Forwardpass aller Beispiele der Minibatch den Optimierungsschritt auszuführen.

Um mit einem der genannten Optimierungsverfahren die optimalen Parameter bezüglich einer Kostenfunktion bestimmen zu können, ist es zunächst notwendig, den Gradienten der Kostenfunktion $\nabla_\theta J$ des Netzwerks möglichst effizient zu berechnen. Dazu eignet sich der Backpropagation Algorithmus, der auf die Arbeit von *Rumelhart et al.* zurückgeht [RHW86]. Der Gradient der Kostenfunktion $\nabla_\theta J$ ist dabei ein Vektor, dessen Komponente mit Index ij die partielle Ableitung der Kostenfunktion bezüglich des Parameters θ_{ij} ist. So ergibt sich die Änderung eines Parameters $\Delta\theta_{ij} = \frac{\partial J}{\partial \theta_{ij}}$, die noch mit der Lernrate η zu verrechnen ist. Nachdem ein Trainingsbeispiel durch das Netz propagiert und der Fehler berechnet wurde, bestimmt der Backpropagation Algorithmus mit Hilfe rekursiver Anwendung der Kettenregel aus der Analysis partielle Ableitungen [GBC16], siehe Formel 2.2.8.

$$\frac{\partial J}{\partial \theta_{ij}} = \frac{\partial J}{\partial o_j} \frac{\partial o_j}{\partial z_j} \frac{\partial z_j}{\partial \theta_{ij}} \tag{2.2.8}$$

Dabei steht $o_j = \phi(z_j)$ für die Ausgabe des Neurons j und $z_j = \sum_{i=1}^{n} x_i w_{ij}$ für die gewichtete Summe der Eingaben x_i des Neurons j.
Befindet sich Neuron j in der letzten Schicht l, ergibt sich $\frac{\partial J}{\partial o_j^l} = \phi'(o_j^l - \hat{y}_j)$, wobei $\hat{y}_j$

hier für die j-te Komponente des Targets steht. Liegt das Neuron j in einer versteckten Schicht l, sind Berechnungen aus der nächsten Schicht zu beachten, siehe Formel 2.2.9

$$\sum_{j=0}^{n_{l+1}-1} w_{jk}^{l+1} \phi'(z_j) \frac{\partial J}{\partial o_j^{l+1}} \tag{2.2.9}$$

2.3 CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNNs) sind künstliche neuronale Netze, die in mindestens einer Schicht die mathematische Faltung (Convolution), eine spezielle lineare Operation, statt allgemeiner Matrixmultiplikation nutzen [GBC16]. Solche Schichten werden Faltungsschichten (Convolutional Layers) genannt und in Kapitel 2.3.1 erläutert. Neben Faltungsschichten und den zugehörigen Aktivierungsfunktionen spielen darauf folgende Pooling Layers (siehe Kapitel 2.3.2) eine wichtige Rolle für CNNs, da sie für Invarianz der Repräsentationen gegenüber kleinen Positionsänderungen (Translations) der Merkmale innerhalb eines Eingabebildes sorgen [GBC16]. Pooling kann als lokales Downsampling betrachtet werden.

Eine der ersten Umsetzungen eines CNNs geht auf *Yann LeCun* [LeC+89] zurück, der das nach ihm benannte LeNet erfolgreich zur Klassifikation von Ziffern einsetzte und damit die Leistungsfähigkeit von CNNs demonstrierte. Zwar stießen frühe CNNs wie das einfach strukturierte LeNet schnell an ihre Leistungsgrenzen, dennoch blieb der Erfolg als bevorzugtes Mustererkennungsverfahren zur Visual Object Recognition dank stetiger Weiterentwicklungen nicht aus. So sorgte die Einführung der ReLU-Aktivierungsfunktion (siehe Kapitel 2.2.2) für die Möglichkeit, CNNs zu konstruieren, die mehr Faltungsschichten als das LeNet enthalten. Ein bekanntes Beispiel für eines der ersten tieferen CNNs ist das AlexNet [KSH12]. Frühe CNNs nutzten die Sigmoid-Aktivierungsfunktion (siehe Kapitel 2.2.2), deren Vanishing Gradient Problem die Nutzung von Architekturen, die eine bestimmte Tiefe aufweisen, unpraktikabel gestaltet.

CNNs haben sich in der Verarbeitung von Daten etabliert, die eine Struktur aufweisen, die sich im euklidischen Raum darstellen lässt. In diese Kategorie fallen beispielsweise zeitreihenbasierte Daten und insbesondere die für Word Spotting essentiellen Word Images, die, wie alle Bilddaten, als zweidimensionale Anordnung von Pixeln betrachtet werden können [GBC16]. Handelt es sich um Farbbilder, so kommt eine weitere Dimension für die Anzahl der Farbkanäle (RGB) hinzu. Ein speziell für Word Spotting konzipiertes CNN, das PHOCNet, wird in Kapitel 3.1 vorgestellt und in der Methodik in Kapitel 4.2 zum DAPHOCNet erweitert.

2.3.1 *Convolutional Layers*

Convolutional Layers (Faltungsschichten) werden eingesetzt, um Merkmale aus Bilddaten zu extrahieren. Die Grundidee dafür lautet, einen Filter, auch Kernel genannt, nach und nach über das Eingabebild I zu schieben und dabei jeweils lokal, innerhalb des sogenannten Receptive Field, mittels Faltung Merkmale zu extrahieren und auf eine sogenannte Feature Map abzubilden, die die Ausgabe einer Faltungsschicht darstellt [GBC16]. Eine Schicht kann mehrere Filter auf die Eingabe I anwenden und somit eine Ausgabe (Feature Maps) produzieren, deren Tiefe der Anzahl der Filter entspricht. Ein Filter ist eine mehrdimensionale Gewichtsmatrix, meistens quadratisch oder kubisch, die im Folgenden mit K bezeichnet wird und dessen Werte lernbare Parameter des Netzes sind. Die hier ausgeführte Rechenoperation, die Faltung (Convolution), wird zunächst in ihrer Grundform in Gleichung 2.3.1 definiert.

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.3.1}$$

Die Funktion $s(t)$ ergibt sich in Gleichung 2.3.1 als gewichteter Mittelwert aus der Gewichtung der Funktion $x(t)$ mit der Gewichtungsfunktion $w(a)$ [GBC16].
Für den Fall, dass das CNN auf zweidimensionalen Bilddaten arbeiten soll, muss die Formel auf den zweidimensionalen Fall verallgemeinert werden, siehe Gleichung 2.3.2.

$$S(i,j) = (I * K)(i,j) = \sum_{m} \sum_{n} I(m,n)K(i-m, j-n) \tag{2.3.2}$$

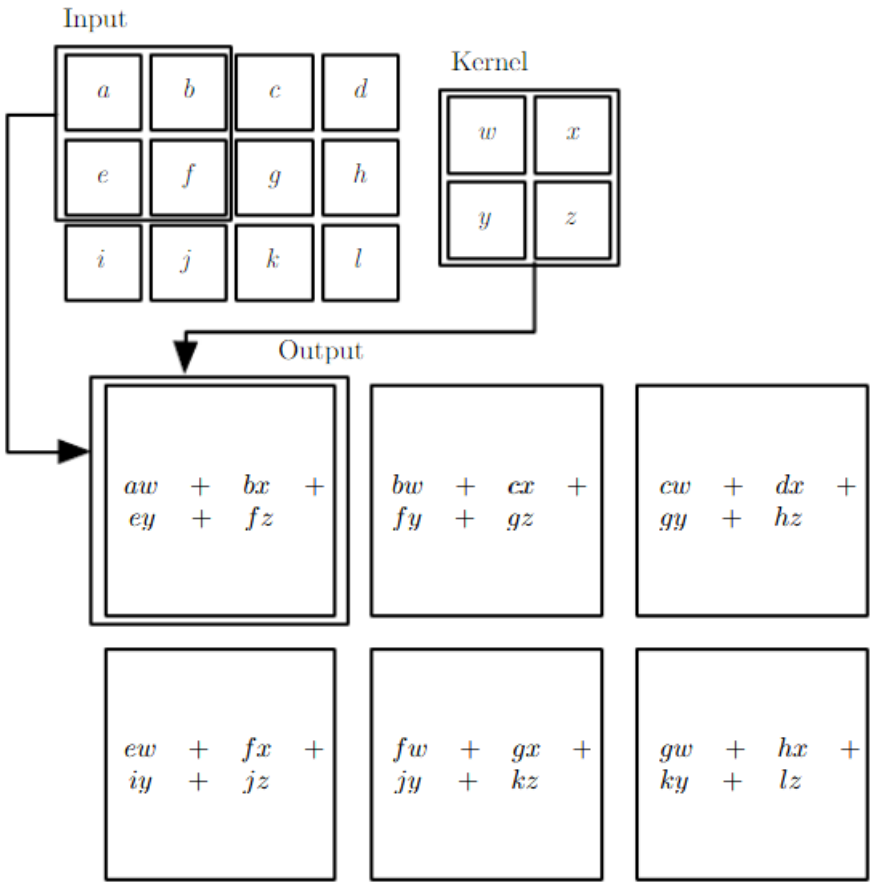


Abbildung 2.3.1: Die Abbildung illustriert eine zweidimensionale Faltung (siehe Gleichung 2.3.2) mit Input I und Kernel K. Dabei werden Patches des Inputs I, hier z.B. [a,b,e,f], die den Ausmaßen des Kernels K in Länge und Breite, hier 2x2, entsprechen, jeweils mit Kernel K gemäß der Gleichung verrechnet. Es findet eine elementweise Multiplikation mit anschließender Aufsummierung der Produkte statt. Die Abbildung ist aus [GBC16] entnommen.

Abbildung 2.3.1 zeigt ein Beispiel für eine zweidimensionale Faltung. In der Praxis wird die Faltung häufig als Kreuzkorrelation implementiert, siehe dazu Gleichung 2.3.3.

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{2.3.3}$$

Extrahierte Merkmale werden von Faltungsschicht zu Faltungsschicht tendenziell komplexer (wenn mehrere Faltungsschichten in einem CNN eingesetzt werden). Die Komplexität der extrahierten Merkmale reicht dabei von anfänglich einfachen geome-

trischen Strukturen, wie horizontalen und vertikalen Kanten, bis hin zu speziellen, komplexen Strukturen, wie beispielsweise die des menschlichen Gesichts.

Faltungsschichten bieten den Vorteil, dass die Filter bzw. Parametermatrizen, für jeden einzelnen Patch der Eingabe I wiederverwendet werden können, was als Parameter Sharing bezeichnet wird [GBC16]. So ist zwar im Sinne der lokalen Konnektivität immer ein Patch mit einem Nachfolgeneuron verknüpft, jedoch können für alle Neuronen, die jeweils mit einem anderen Teilbereich der Eingabe verknüpft sind, dieselben Gewichte verwendet werden. Das bedeutet, dass Neuronen einer Schicht sich einen Filter teilen. Dies gilt auch, wenn mehrere Filter verwendet werden, um verschiedene Features zu extrahieren. Ist es sinnvoll, ein bestimmtes Merkmal an einer Stelle im Raum zu berechnen, so ist es sicherlich auch an einer anderen Stelle nützlich [LKX]. Aufgrund des Parameter Sharings müssen im CNN deutlich weniger Informationen zu Parametern gespeichert werden als in normalen KNNs [GBC16]. Faltung führt daher auch zu einer höheren Effizienz gegenüber der üblichen Matrizenmultiplikation in vollvernetzten Schichten.

Zusätzlich ermöglicht Faltung das Arbeiten mit Eingaben variabler Größe, was zur Folge hat, dass Eingabebilder nicht auf dieselbe Größe gebracht werden müssen [GBC16]. Das ist ein Vorteil gegenüber herkömmlichen KNNs, die eine feste Eingabegröße voraussetzen, da durch Zurechtstutzten und Skalieren von Bildern eventuell Informationen verloren gehen. Wird nach den Faltungsschichten ein MLP zur Klassifikation angeschlossen, ist es jedoch nötig, die Größe der produzierten Feature Maps auf eine feste Größe herunterzubrechen. Dies kann beispielsweise mittels Spatial Pyramid Pooling (siehe Kapitel 2.3.2) umgesetzt werden.

2.3.2 *Pooling Layers*

Convolutional Layers (siehe Kapitel 2.3.1) bilden zusammen mit darauffolgenden Aktivierungsfunktionen nur einen von zwei Grundbausteinen, aus denen ein CNN für gewöhnlich aufgebaut ist. Nachdem in einer Faltungsschicht Faltungen und anschließende Aktivierungen durchgeführt worden sind, besteht die Möglichkeit, die so berechneten Feature Maps mit einem Pooling Layer noch weiter zu manipulieren [GBC16]. Pooling löst das Problem, dass Features in der Ausgabe einer Faltungsschicht, der Feature Map, präzise dem Ort ihrer Entnahme aus der Eingabe zugeordnet werden. Eine kleine Änderung an den Merkmalen des Eingabebilds, beispielsweise durch Rotation oder Translation, führt so ohne Pooling zu einer anderen Feature Map, obwohl das Bild insgesamt noch dieselben Merkmale enthalten kann, eventuell nur an leicht verschiedenen Positionen. Pooling sorgt dafür, dass Repräsentationen von Eingaben nahezu invariant gegenüber kleinen Veränderungen werden [GBC16]. Des Weiteren wird mittels Pooling die Anzahl der Parameter verringert, was zu schnelleren Berechnungen führt [LKX].
Diese Eigenschaften werden dadurch erreicht, dass Ausgaben eines Convolutional Layers durch statistische Zusammenfassungen ihrer Nachbarn ersetzt werden. Mit Ausgaben sind hier Teile (Patches) von Feature Maps gemeint. Dieses Vorgehen wird auch als Downsampling bezeichnet, da Pooling die Größe einer Repräsentation verringert. Abbildung 2.3.2 zeigt mit Maximum Pooling (Max Pooling) [ZC88] eine der Varianten statistischer Zusammenfassung.

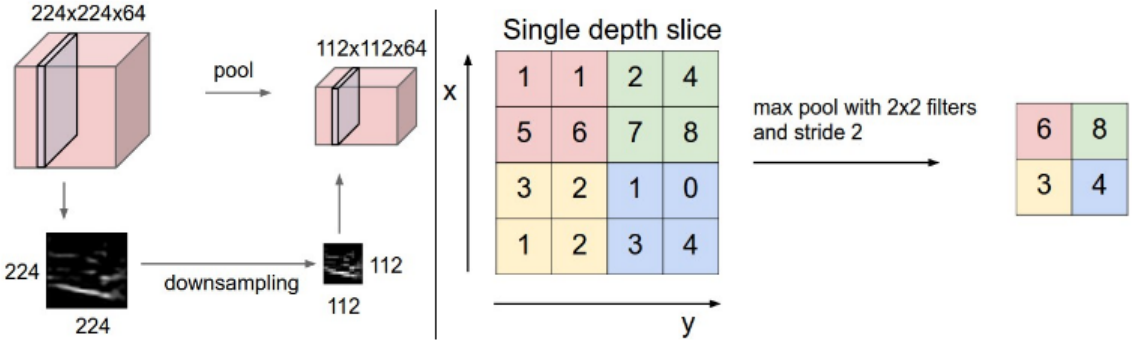


Abbildung 2.3.2: Das aus 64 Feature Maps (hier auf der linken Seite als Slices zu sehen) bestehende Eingabevolumen wird mittels Max Pooling von der Größe $224 \times 224 \times 64$ auf die Größe $112 \times 112 \times 64$ reduziert. Die Anzahl der Feature Maps bleibt unverändert. Das Downsampling der einzelnen Feature Maps mittels Max Pooling findet mit 2×2 Filtern und einer Schrittweite (Stride) von 2 statt (siehe rechte Seite). Aus jedem 2×2 Patch (rot, grün, gelb und blau) wird der maximale Wert abgebildet. Die Abbildung ist aus [LKX] entnommen.

Max Pooling ist neben weiteren Funktionen wie Average Pooling die meistverwendete Pooling Variante [LKX] und findet auch im PHOCNet (Kapitel 3.1), sowie im DAPHOCNet (Kapitel 4.2) Anwendung.

Spatial Pyramid Pooling Layer

He et al. [He+14] präsentierten 2015 einen speziellen Pooling Layer, der auf Spatial Pyramid Pooling [GD05][LSP06] basiert. Der Spatial Pyramid Pooling Layer soll das Problem lösen, dass nur Eingabebilder fester Größe in Netzarchitekturen verwendet werden können, die aus einem Konvolutionsteil und einem nachfolgenden, vollvernetzten Teil bestehen. Der Grund für den Zwang zu Eingaben fester Größe liegt nicht an den Convolutional Layers, die Eingaben beliebiger Größe verarbeiten und Feature Maps verschiedener Größen generieren können, sondern am angeschlossenen MLP, das zur Klassifikation der im Konvolutionsteil extrahierten Merkmale verwendet wird und in Form der ersten vollvernetzten Schicht eine fest definierte Eingabegröße aufweist. Die üblicherweise verwendete Methode, um zu gewährleisten, dass Feature Maps aus dem Konvolutionsteil des Netzes zur ersten vollvernetzten Schicht des MLP passen, beschränkte sich auf das Zurechtschneiden (Cropping) oder Skalieren (Warping) der Eingabebilder.

Abbildung 2.3.3 visualisiert im oberen Teil, wie Bilder mittels Cropping und Warping auf eine feste Größe gebracht werden können. Bilder, die mit solchen Verfahren auf eine feste Größe gebracht werden, enthalten oft nicht mehr alle Informationen, die zu akkurater Erkennung nötig gewesen wären. So enthält beispielsweise die Bildregion, die aus dem Cropping resultiert (zweites Bild v. l.), nicht mehr das ganze Fahrzeug (erstes Bild v. l.).

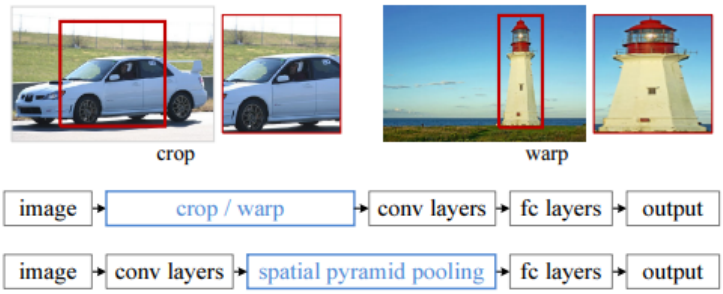


Abbildung 2.3.3: **Mitte:** Schematische Darstellung einer normalen CNN-Pipeline mit Änderung der Größe der Eingabe (crop/warp).
Unten: Schematische Darstellung einer CNN-Pipeline, die keine Größenänderung der Eingabebilder beinhaltet, sondern stattdessen einen SPP-Layer, der auf die letzte Faltungsschicht folgt und Repräsentationen fester Größe produziert, die anschließend in die Fc-Layers geleitet werden. Die Abbildung ist aus [He+14] entnommen.

Das Grundprinzip lautet, zwischen dem letzten Convolutional Layer und dem ersten Fully-connected Layer den SPP-Layer einzusetzen, der die bisher generierte Repräsentation so komprimiert, dass sich auf Basis der gegebenen Hyperparameter eine Repräsentation fester Länge ergibt [He+14]. Auf diese Weise werden Methoden zur Größenänderung wie Cropping und Warping für die CNN-Pipeline irrelevant.

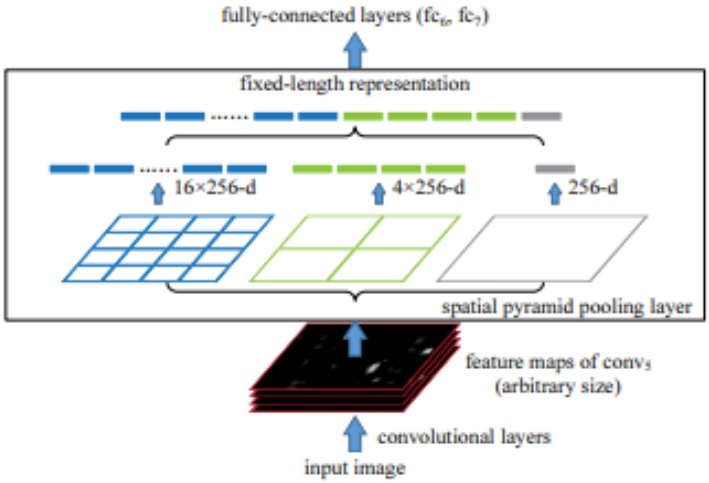


Abbildung 2.3.4: Visualisierung der Position des SPP-Layers im Gesamtnetz als auch der Funktionsweise des Spatial Pyramid Pooling Ansatzes. Die Inhalte der 256 Feature Maps werden jeweils in $16+4+1$ Spatial Bins gepoolt. Die Abbildung ist aus [He+14] entnommen.

Die in Abbildung 2.3.4 dargestellte Funktionsweise des SPP-Layers zeigt, dass Informationen aus den Feature Maps in sogenannte Spatial Bins eingeordnet werden. Dabei spielt die Größe der Feature Maps keine Rolle, da die Spatial Bins sich in ihrer Größe proportional zur Größe der Feature Maps verhalten [He+14]. Die Anzahl der Spatial Bins ist jedoch fest, um unabhängig von der Eingabegröße eine feste Länge der resultierenden Repräsentation sicherzustellen. Das Pooling der in den Feature Maps gespeicherten Informationen in die Spatial Bins bewahrt die räumlichen Informationen. Die Ausgaben des Spatial Pyramid Pooling sind k M-dimensionale Vektoren, wobei M für die Anzahl der Spatial Bins steht und k die Anzahl der Filter in der letzten Konvolutionsschicht angibt [He+14]. Diese k M-dimensionalen Vektoren bilden die Eingabe der vollvernetzten Schicht, die auf den Konvolutionsteil und das Spatial Pyramid Pooling folgen.

Temporal Pyramid Pooling Layer

Eine spezielle Variante des Spatial Pyramid Poolings ist das Temporal Pyramid Pooling (TPP), das sich durch die Aufteilung von Feature Maps in feinere Bereiche entlang der

horizontalen Achse auszeichnet, aus denen Features gepoolt werden [SF18]. Im Gegensatz zu klassischem SPP (siehe Abbildung 2.3.4) werden Features der Repräsentationen nicht aus einer gleichmäßigen Partition entlang der horizontalen und vertikalen Achse in Spatial Bins gepoolt. Diese Art der Aufteilung, die den Fokus auf die horizontale Achse legt, eignet sich insbesondere für Wortabbilder, da es sich um Abbildungen von Zeichenketten, das heißt sequentielle Daten handelt, bei denen die Anordnung der Zeichen von links nach rechts eine wichtige Rolle spielt. Jede Zelle, die durch eine Aufteilung eines Wortabbildes entlang der Schreibrichtung entsteht, repräsentiert daher Merkmale aus aufeinanderfolgenden Intervallen des Wortabbildes [SF18]. Die Verkettung der mittels Pooling aus Zellen verschiedener Aufteilungen (Levels) entlang der Schreibrichtung entstandenen Repräsentationen ergibt wie beim SPP-Ansatz eine Gesamtrepräsentation fester Länge, die aber zusätzlich den Schreibverlauf einbezieht.

2.4 LERNPARADIGMEN

In Kapitel 2.2.3 wurde das Konzept der Kostenfunktion eingeführt, das die Berechnung eines Fehlers aus Approximation und Target umfasst. Sind Trainingsziele vorgegebenen, können mit Hilfe der Kostenfunktion Fehler berechnet und damit die Approximation der zu lernenden Funktion verbessert werden, indem die Kostenfunktion minimiert wird. Es wird also versucht auf Basis gegebener Datenpunkt-Label-Paare den Funktionswert einer unbekannten Funktion für neue Datenpunkte zu approximieren [RN09][LT15]. In diesem Fall wird von Supervised Learning (überwachtem Lernen) gesprochen [LT15]. Überwachtes Lernen wird im PHOCNet (siehe Kapitel 3.1) und DAPHOCNet (siehe Kapitel 4.2) zur Approximation von PHOC-Attributvektoren genutzt.

Ein großer Nachteil überwachter Lernverfahren ist der Bedarf an annotierten Trainingsdaten, da diese teuer und schwer zu erlangen sein können. In vielen Fällen müssen Annotationen manuell erstellt werden, was dafür sorgt, dass es oft schwer ist, annotierte Trainingsdaten anzusammeln, die zur Zieldomäne (Testdaten) passen [WKW16]. Ein Lösungsansatz dafür lautet, Trainingsdaten aus einer anderen, aber ähnlichen Domäne zu entnehmen, für die Annotationen leichter zu erlangen sind. Die Grundidee dieses Verfahrens ist es daher, das Wissen, das mit Hilfe von Daten aus einer ähnlichen Domäne erlernt wurde, auf eine andere Domäne, auch Zieldomäne genannt, zu transferieren. Diese Methodik wird als Transfer Learning bezeichnet [WKW16].

Zur Abgrenzung der einzelnen Lernverfahren ist der Begriff Überwachung essentiell, da sich maschinelle Lernverfahren am Grad der Überwachung unterscheiden lassen. Mit Überwachung ist in diesem Kontext der Einfluss eines Menschen auf das Training eines Modells gemeint. Anders formuliert lassen sich Lernverfahren über die Menge an Folgerungen abgrenzen, die ein Verfahren aus den von Experten vorgegebenen

Informationen zieht [LT15]. Werden außer den Trainingsdaten keine Informationen mitgegeben, können daraus auch keine Schlüsse gezogen werden und folglich handelt es sich um Unsupervised Learning (unüberwachtes Lernen). Das System muss also auf Basis eines Ähnlichkeitskriteriums Datenpunkte gruppieren und schließlich ein verbindendes Konzept folgern [LT15].

Neben überwachtem und unüberwachtem Lernen gibt es weitere Abstufungen im Bezug auf den Einfluss eines Experten auf das Training. Sogenanntes Semi-supervised Learning [ZG09] umfasst das Lernen mit Trainingsdaten, für die nur teilweise Label vorhanden sind. Eine noch schwächere Variante wird als Weak Supervision (schwache Überwachung) bezeichnet [SGF17]. Bei Weak Supervision gibt der Experte gelabelte Trainingsdaten in geringem Umfang vor.

Ein weiterer Bereich der maschinellen Lernverfahren ist Reinforcement Learning (verstärkendes Lernen). Hierbei gibt es keine Datenpunkt-Zielwert-Paare [LT15]. Es geht um die Ausführung von Aktionen durch einen lernenden Agenten. Sind ausgeführte Aktionen gewünscht und somit korrekt, wird der Agent belohnt. Andernfalls erhält er ein Reinforcement Signal [LT15], sodass eine derartige Aktion möglichst nicht erneut ausgeführt wird.

3 VERWANDTE ARBEITEN

3.1 PHOCNET

Das PHOCNet [SF16] ist ein speziell für Word Spotting entwickeltes CNN, das PHOCs (**P**yramidal **H**istogram **O**f **C**haracters) für Eingaben berechnet. PHOCs sind Attributvektoren, wobei Attribute hier für einzelne Zeichen in spezifischen Bereichen des Bildes, sogenannten Splits, stehen. Sie werden im Kapitel PHOC näher betrachtet. Die Verwendung des Ansatzes eines gemeinsamen Attributraums [Alm+14] für Text- und Wortabbildrepräsentationen befähigt das PHOCNet dazu, sowohl QbE, als auch QbS durchzuführen [SF18]. Für Texteingaben (QbS) ist es trivial, die Attributrepräsentation zu bestimmen (siehe Abbildung 3.1.1). Liegt ein Wortabbild als Eingabe vor (QbE), approximiert das PHOCNet die Attributrepräsentation. Insgesamt sorgt die Projektion von Wortabbildern und Strings in einen gemeinsamen PHOC-Attributraum dafür, dass sich die Ähnlichkeit von Anfragen und Wortabbildern eines Datensatzes leicht mit Hilfe von Distanzvergleichsmetriken berechnen lässt [SF18]. Dazu kann beispielsweise die Kosinusdistanz verwendet werden, siehe Gleichung 3.1.1:

$$d_{cos}(\mathbf{a}, \mathbf{b}) = 1 - \frac{\mathbf{a}^T \mathbf{b}}{\|\mathbf{a}\| \cdot \|\mathbf{b}\|} \tag{3.1.1}$$

Ein alternativer Ähnlichkeitsvergleich kann mittels Einbettung in ein probabilistisches Modell [Rus+18a] durchgeführt werden .

Die Besonderheiten des PHOCNets im Vergleich zu üblichen CNNs äußern sich vorallem in zwei Aspekten. Zum einen gibt es den in Kapitel 2.3.2 eingeführten Spatial Pyramid Pooling Layer, der es CNNs ermöglicht, Eingabebilder verschiedener Größen zu verarbeiten. Nachteile, die sich sonst durch Bildmanipulation ergeben, entfallen. Zum anderen verzichtet die PHOCNet-Architektur auf die typischerweise verwendete Softmax-Aktivierungsfunktion und nutzt stattdessen eine Sigmoid-Aktivierungsfunktion zur Klassifikation [SF18]. Die genaue Einordnung dieser Aspekte in die PHOCNet-Architektur, sowie weitere Details werden im Kapitel Netzarchitektur vorgestellt.

PHOC

Ein **P**yramidal **H**istogram **O**f **C**haracters ist eine von *Almazán et al.* [Alm+14] entwickelte Attributrepräsentation und wird auch als eine binäre, pyramidale Repräsentation einer Zeichenkette verstanden [SF16]. Attribute sind in diesem Kontext visuelle Merkmale, die sich mehrere Wortabbilder teilen können. Intuitiv entsprechen die Attribute eines Wortabbildes den vorkommenden Zeichen des Wortes [SF16]. Es wird von einer pyramidalen Repräsentation gesprochen, da PHOCs kodieren, ob ein Attribut in bestimmten Splits, das bedeutet Aufteilungen der Zeichenkette, vorkommt.

Abbildung 3.1.1 illustriert, wie durch verschiedene Aufteilungen der Zeichenkette verschieden lange Repräsentationen entstehen, die hinterher alle miteinander verkettet werden, um ein PHOC zu ergeben. Die Aufteilung auf Level 1 betrachtet die gesamte Zeichenkette (globale Aufteilung) und hat daher nur eine Attributkodierung. Wird ein String nun wie auf Level 2 in der Mitte aufgeteilt, wird für beide Hälften die vollständige Attributkodierung über alle definierten Zeichen angelegt. Level 3 zeigt eine Aufteilung in drei Teilstrings.

Mit dem PHOC kann Wissen über Attribute aus Trainingsbeispielen auf Testdaten übertragen werden, insofern alle Attribute, die in den Testdaten vorkommen, vorher in den Trainingsdaten enthalten waren [SF16].

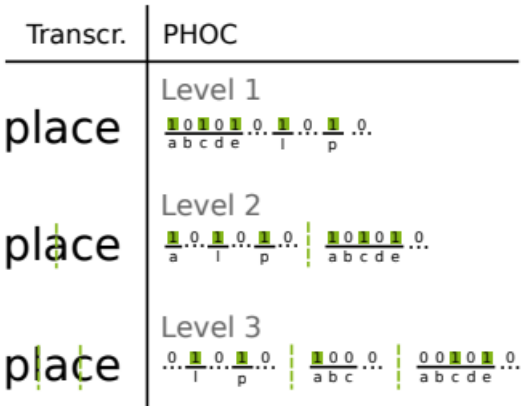


Abbildung 3.1.1: PHOC Extraktion bei gegebenem Textstring. Die Abbildung ist aus [SF18] entnommen.

Netzarchitektur

In Abbildung 3.1.2 ist die Architektur des PHOCNets dargestellt. Im Konvolutionsteil des PHOCNets werden 3×3-Filter verwendet, da diese verglichen mit größeren Filtern bessere Ergebnisse liefern [SZ14]. Auf die Konvolutionen folgen jeweils ReLUs. Entsprechend der Architektur in [SZ14] arbeiten tiefer im Netz liegende Konvolutionsschichten mit mehr Filtern als jene, die weiter vorne liegen, um eine größere Anzahl komplexer Merkmale zu lernen [SF16].

Auf den Konvolutionsteil des PHOCNets folgt ein Spatial Pyramid Pooling Layer, der es ermöglicht, Feature Maps beliebiger Größe auf eine fixe Repräsentationsgröße abzubilden. Das ist wichtig, weil so die inhärente Eigenschaft von Konvolutionsschichten, Eingabebilder verschiedener Größen verarbeiten zu können, tatsächlich genutzt werden kann, ohne dass diese zunächst angepasst werden müssen, damit später berechnete Repräsentationen zur ersten vollvernetzten Schicht des PHOCNets passen. Die vom SPP-Layer erzeugten Repräsentationen werden dann als Eingabe des MLPs genutzt, das dem vollvernetzten Teil des PHOCNets entspricht. In der letzten Schicht des PHOCNets wird die Sigmoid-Aktivierungsfunktion auf die Ausgaben der letzten vollvernetzten Schicht angewendet, die sich zur Multi-Label Klassifikation eignet [SF18]. Im Kontext von PHOCs wird von einer Multi-Label Klassifikation gesprochen, da mehrere Elemente eines PHOCs den Wert 1 annehmen können und sich die Softmax-Aktivierungsfunktion dafür nicht eignet [SF18].

In der Methodik dieser Arbeit wird eine leicht modifizierte Version des PHOCNets, das sogenannte TPP-PHOCNet [SF18], verwendet. Der Unterschied zum klassischen PHOCNet liegt in der Nutzung einer speziellen Variante des allgemeinen SPP-Layers, dem Temporal Pyramid Pooling Layer (TPP) (siehe Kapitel 2.3.2). Die Nutzung eines TPP-Layers anstelle des klassischen SPP-Layers hat im Bereich der Document Image Analysis zu besseren Resultaten geführt [SF18].

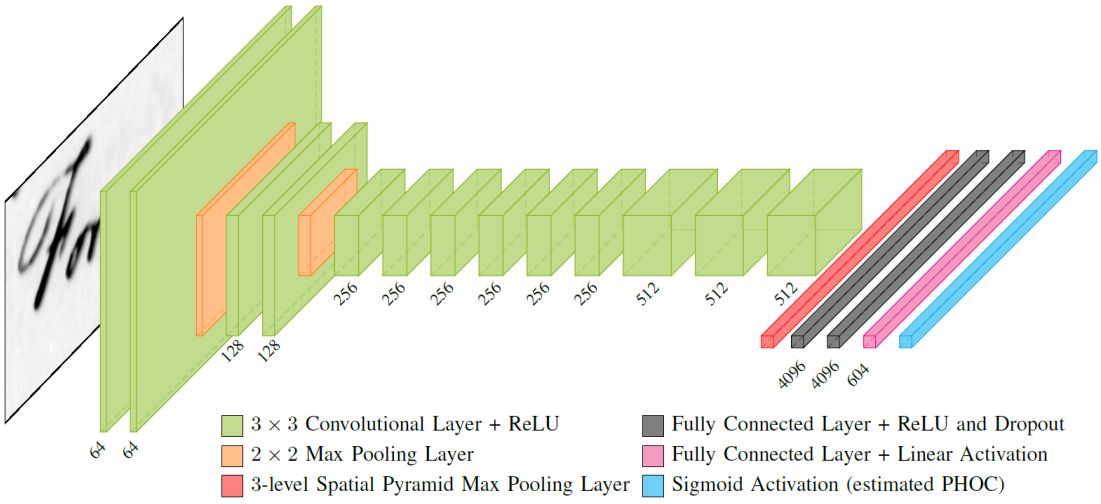


Abbildung 3.1.2: Illustration des Aufbaus des PHOCNets, das sich als Kombination vier essentieller Grundbausteine beschreiben lässt: Die grün gefärbten Konvolutionsschichten ergeben zusammen mit den orangenen Pooling Layers und ReLUs den Konvolutionsteil des Netzes, der Feature Maps berechnet. Es folgt ein 3-Level SPP-Layer in Rot und schließlich auf den vollvernetzten Teil die Sigmoid-Aktivierung, die als blaue Box dargestellt ist. Die Zahlen unterhalb der Konvolutionsschichten geben die Anzahl der verwendeten Filter an. Die Neuronenanzahl der vollvernetzten Schichten ist dementsprechend jeweils unter den dazu korrespondierenden Boxen angegeben. In der vorletzten Schicht entspricht die Anzahl der Neuronen der Größe des PHOCs. Die Abbildung ist aus [SF16] entnommen.

3.2 DOMAIN ADAPTATION

Annotierte Trainingsdaten für CNNs in großen Mengen zu erstellen ist kostspielig und mühsam. Synthetische Trainingsdaten (siehe Kapitel 5.1.1) werden daher insbesondere in der Word Spotting und HTR (Handwritten Text Recognition) Community zunehmend beliebter. Sie weisen jedoch das Problem auf, dass extrahierte Merkmale sich merklich von denen realer Trainings- und Testdaten unterscheiden [Kan+20]. Ein Domänenwechsel von synthetischen Trainingsbeispielen auf reale Testdaten kann daher zu vergleichsweise schlechten Ergebnissen führen. Domain Adaptation kann dem entgegenwirken, indem Quell- und Zieldomäne einander angenähert werden. *Kang et al.* [Kan+20] stellten in diesem Kontext kürzlich einen Ansatz für den HTR Bereich vor, bei dem ein auf synthetischen Daten trainiertes CNN unüberwacht an eine Zieldomäne angepasst wird. HTR gehört zu den erkennungsbasierten Verfahren, die genutzt werden, um Transkriptionen von Zeichenketten zu berechnen oder Anfragen zu klassifizieren [Gio+17].

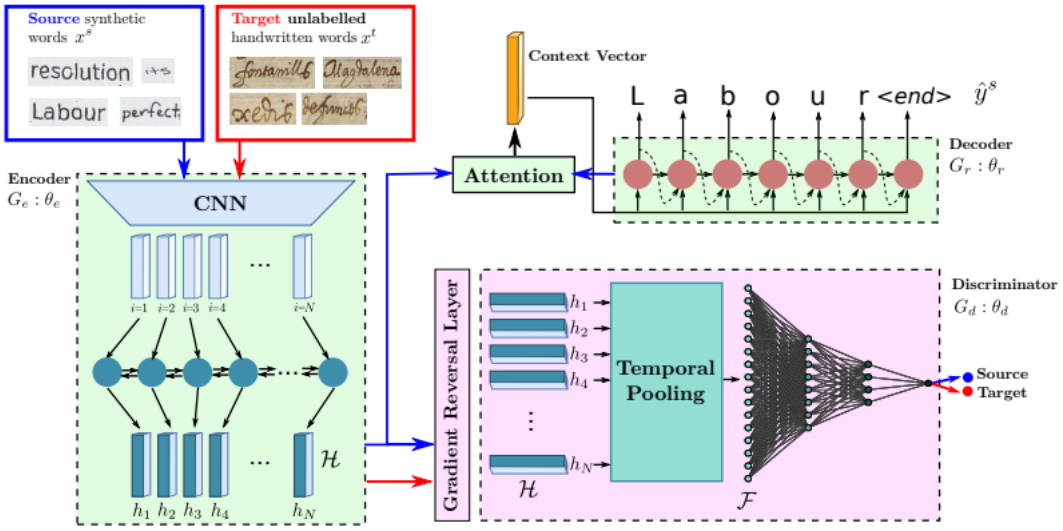


Abbildung 3.2.1: Visualisierung des Domain Adaptation Ansatzes von *Kang et al.* Das grüne Segment, bestehend aus Endcoder und Decoder, stellt den Recognizer des Ansatzes dar. Der Diskriminator entspricht der pinken Komponente des Ansatzes. Beispiele aus der Quelldomäne (Source) fließen während des Trainings durch beide Komponenten, die realen, nicht annotierten Daten nehmen nur Einfluss auf das Training des Endcoders und des Diskriminators. Die Abbildung ist aus [Kan+20] entnommen.

Abbildung 3.2.1 zeigt die Architektur des von *Kang et al.* vorgestellten Ansatzes, der aus drei Grundbausteinen besteht. Der Encoder genannte Teil ist als CNN wie der Konvolutionsteil des PHOCNets für die Merkmalsextraktion zuständig. Der Diskriminator ist der für die Domain Adaptation essentielle Teil der Architektur. Es handelt sich dabei um einen Binärklassifikator, der zwischen Eingaben aus Quell- und Zieldomäne unterscheidet. Um das Ziel des Ansatzes zu erreichen, das heißt, das Recognition Model so anzupassen, dass es eine gute Leistungsfähigkeit auf Quell- und Zieldomäne hat, wird die Strategie Adversarial Learning [Gan+16] genutzt. Die Idee lautet dabei, die Diskrepanz zwischen zwei verschiedenen Merkmalsverteilungen zu minimieren, indem zwei Komponenten eines Netzwerks, hier Recognizer und Diskriminator, gemeinsam auf eine gegensätzliche Weise trainiert werden. Der entscheidene Punkt dieses Trainingskonzeptes ist es, den Fehler des Recognizers, L_r, zu minimieren, während der Fehler des Diskriminators, L_d, maximiert wird [Kan+20]. Intuitiv sollen domäneninvariante Features gelernt werden, die für den Diskriminator nicht unterscheidbar sind. Die Maximierung des Diskriminatorloss L_d geschieht mit Hilfe des Gradient Reversal Layers (GRL), der das Vorzeichen des Gradienten aus dem Diskriminatorteil umdreht, sodass der negative Fehler an den Endcoder mittels Backpropagation zurückgeleitet wird. Auf diese Weise lässt sich das Netzwerk in einem

min-max Stil trainieren. Formel 3.2.1 definiert den für das gesamte Netz geltenden Loss

$$L(\theta_e, \theta_r, \theta_d) = \sum_{x_i \in D_s} L_r(G_r(G_e(x_i)), y_i) - \lambda \sum_{x_j \in D_s \cup D_t} L_d(G_d(G_e(x_j)), d_j). \quad (3.2.1)$$

$G_e(x)$, $G_r(x)$ und $G_d(x)$ stehen für die Ausgaben des jeweiligen Netzwerkteils und D_s und D_t für Daten aus Source- und Targetdomain (Quell- und Zieldomäne). Das λ ist ein Faktor, mit dem sich der Einfluss des Diskriminators auf den Endcoder regulieren lässt [Kan+20]. Das Optimierungsziel lautet, einen Sattelpunkt zu finden, sodass

$$\hat{\theta}_e, \hat{\theta}_r, = \underset{\theta_e, \theta_r}{\operatorname{argmin}} L(\theta_e, \theta_r, \theta_d) \quad (3.2.2)$$

$$\hat{\theta}_d = \underset{\theta_d}{\operatorname{argmax}} L(\theta_e, \theta_r, \theta_d). \quad (3.2.3)$$

Ein Punkt, wo sich der Ansatz von *Kang et al.* von anderen Adversarial Learning Ansätzen mit Domain Adaptation [GL15] unterscheidet, ist die Nutzung eines Temporal Pooling Layers im Diskriminatorteil. Da hier Text verarbeitet wird, das heißt sequentielle Signale variabler Länge, ist es nötig, die variable Merkmalsrepräsentation auf eine fixe Repräsentation abzubilden [Kan+20]. Der Ansatz von *Ganin und Lempitsky* [GL15] weist dieses Problem nicht auf, da dort MNIST, MNIST-M und SVHN als Datensätze verwendet werden. Die korrespondierenden Repräsentationen sind nicht von variabler Länge. Abbildung 3.2.2 visualisiert die Architektur des Ansatzes von *Ganin und Lempitsky*, die eine große strukturelle Ähnlichkeit zum zuvor diskutierten Ansatz aufweist. Auch hier gibt es drei Teilstrukturen, in die das Netzwerk aufgeteilt werden kann: Der Konvolutionsteil zur Merkmalsextraktion, ein Binärklassifikator (Diskriminator) und ein Klassifikator für die zu approximierenden Ziffernabbildungen.

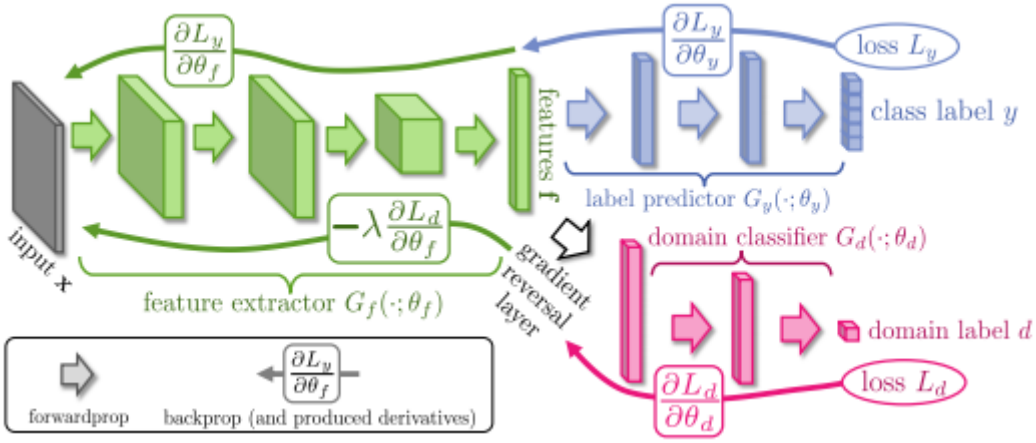


Abbildung 3.2.2: Visualisierung des Domain Adaptation Ansatzes von *Ganin und Lempitsky*. Der Konvolutionsteil ist grün markiert und der vollvernetzte Bereich zur Labelapproximation blau. Der zur Domain Adaptation notwendige Diskriminator (auch: Domain Classifier) wird durch die magentafarbene Komponente dargestellt. Auch hier wird der Gradient des Domain Classifiers mit einem negativen Faktor λ multipliziert, um das Netz im min-max Stil zu trainieren. Während des Forwardpass berechnet der Gradient Reversal Layer die Identitätsfunktion. Es wird nur während des Backwardpass das Vorzeichen umgedreht. Die Abbildung ist aus [GL15] entnommen.

4 METHODIK

Im Fokus der Methodik dieser Arbeit steht die Verwendung synthetischer Trainingsdaten im Kontext von CNNs, die auf Word Spotting ausgelegt sind. Synthetische Trainingsdaten stellen eine kostengünstige Alternative zu realen Trainingsdaten dar, weil sie relativ einfach zu erstellen sind und manueller Annotationsaufwand entfällt. Die Nutzung synthetischer Trainingsdaten ist trotz dieses immensen Vorteils nicht unproblematisch, da sich extrahierte Merkmale synthetischer Wortabbilder von extrahierten Merkmalen realer Wortabbilder erkennbar unterscheiden. Die Methodik dieser Arbeit geht diese Problematik an, indem sie den Schwerpunkt nacheinander auf folgende Aspekte legt: Zunächst werden drei verschiedene CNN-Architekturen diskutiert, die auf die Approximation von PHOCs zugeschnitten werden. Diese CNNs realisieren die Abbildung von Wortabbild auf Attributrepräsentation (PHOC - siehe Kapitel 3.1). Bei den drei Netzarchitekturen handelt es sich um das LeNet [LeC+98], das PHOCNet [SF18] und das ResNet [He+16]. Alle drei Architekturen werden in Anlehnung an die Arbeit von *Rusakov et al.* [Rus+18b] modifiziert. Das LeNet und das ResNet werden in Kapitel 4.1 so angepasst, dass sie PHOCs approximieren können. Im PHOCNet wird der SPP-Layer durch einen TPP-Layer (siehe Kapitel 2.3.2) ersetzt. Im Mittelpunkt der Untersuchung dieser CNNs steht die Frage, wie sich die Leistungsfähigkeit der drei Netzarchitekturen bei einem Domänenwechsel von synthetischen Trainingsdaten auf reale Testdaten verhält. Im zweiten Teil der Methodik wird das im Word Spotting etablierte TPP-PHOCNet so erweitert, dass es Domain Adaptation durchführen kann. Dieses spezielle CNN wird in Kapitel 4.2 als DAPHOCNet eingeführt. Das DAPHOCNet soll eine synthetische Merkmalsdomäne an eine neue, reale Merkmalsdomäne anpassen. In der Auswertung wird mit dem DAPHOCNet untersucht, inwiefern sich die Baselineresultate für reale Testdaten bei Nutzung synthetischer Trainingsdaten durch Domain Adaptation verbessern lassen.

4.1 NETZARCHITEKTUREN

Nachfolgend werden drei Netzarchitekturen beschrieben, die sich durch Modifikationen leicht von ihren ursprünglichen Formen unterscheiden. Zwei der Netzarchitekturen, das LeNet und das ResNet, müssen für das Word Spotting mit Attributvektoren angepasst werden. Die andere Architektur, das PHOCNet, dient als etablierte Architektur zum Vergleich. Die Komplexität der Architekturen steigt dabei in der Reihenfolge ihrer Nennung.

PHOCLeNet

Das LeNet ist das CNN mit der geringsten Komplexität von den hier vorgestellten Architekturen. Es war das erste CNN, das überwacht mit dem Backpropagation Algorithmus trainiert wurde [LeC+98]. Im Vergleich zu moderneren CNNs weist der Konvolutionsteil des LeNets nur eine geringe Tiefe auf. Insgesamt vier Schichten, davon jeweils zwei Kombinationen einer Konvolutionsschicht und einer Max Pooling Schicht, werden im LeNet verwendet. Die erste Konvolutionsschicht arbeitet mit 28×28-Filtern und berechnet damit 6 Feature Maps. Die zweite Konvolutionsschicht nutzt 16×16-Filter und generiert 16 Feature Maps. Der vollvernetzte Teil des LeNets umfasst zwei verdeckte Schichten mit 120 respektive 84 Neuronen. Die Anzahl der Neuronen der Ausgabeschicht wird durch die Anzahl der zu approximierenden Klassen vorgegeben. Alle verdeckten Schichten des LeNets, insbesondere auch die Konvolutionsschichten, verwenden die Hyperbolic Tangent Aktivierungsfunktion [LeC+98]. Ohne Weiteres ist dieses ursprüngliche LeNet nicht für die Approximation von PHOCs im Kontext des Word Spottings zu gebrauchen. *Rusakov et al.* [Rus+18b] schlagen daher drei Änderungen vor, aus denen das sogenannte PHOCLeNet resultiert. Zunächst werden die Hyperbolic Tangent Aktivierungsfunktionen durch ReLUs ersetzt, die die Trainingszeit verringern und Vanishing Gradients verhindern. Damit das entstehende PHOCLeNet Eingabewortabbilder beliebiger Größe verarbeiten kann, wird die zweite Max Pooling Schicht gegen eine Temporal Pyramid Pooling Schicht (TPP) ausgetauscht. Die letzte Änderung bezieht sich auf die Anzahl der Parameter und Filter. Die Zahl der Filter wird von 6 und 16 auf 20 und 50 erhöht, was das Erlernen einer größeren Anzahl von Merkmalen ermöglicht. Die zwei vollvernetzten Schichten, die im LeNet aus 120 und 84 Neuronen bestehen, werden beide auf 500 Neuronen erweitert. Die Neuronenanzahl der Ausgabeschicht entspricht der Anzahl der Attribute. Die Schichten des PHOCLeNets werden in Tabelle 4.1.1 (linke Spalte) zusammengefasst.

Layer	PHOCLeNet	TPP-PHOCNet	PHOCResNet
Convolution	28 x 28 conv, stride 1 out_channels 20	2 · (3 x 3 conv), stride 1 out_channels 64	7 x 7 conv, stride 2 out_channels 64
Pooling	2 x 2 maxpool, stride 2 out_channels 20	2 x 2 maxpool ,stride 2 out_channels 64	3 x 3 maxpool, stride 2
Convolution	14 x 14 conv, stride 1 out_channels 50	2 · (3 x 3 conv), stride 1 out_channels 128	$\begin{bmatrix} 1x1, 64 \\ 3x3, 64 \\ 1x1, 256 \end{bmatrix} \cdot 3$
Pooling		2 x 2 maxpool, stride 2 out_channels 128	
Convolution		6 · (3 x 3 conv), stride 1 out_channels 256	$\begin{bmatrix} 1x1, 128 \\ 3x3, 128 \\ 1x1, 512 \end{bmatrix} \cdot 4$
Convolution		3 · (3 x 3 conv), stride 1 out_channels 512	$\begin{bmatrix} 1x1, 256 \\ 3x3, 256 \\ 1x1, 1024 \end{bmatrix} \cdot 6$
Convolution			$\begin{bmatrix} 1x1, 512 \\ 3x3, 512 \\ 1x1, 2048 \end{bmatrix} \cdot 3$
Pooling	3 Level TPP-Layer		
MLP	500 fc, ReLU	4096 fc, ReLU, Dropout	
	500 fc, ReLU	4096 fc, ReLU, Dropout	
	604 fc, Sigmoid		

Tabelle 4.1.1: Die Schichten des PHOCLeNets, TPP-PHOCNets und PHOCResNets im Überblick.

TPP-PHOCNet

Das in Kapitel 3.1 vorgestellte PHOCNet [SF18] ist speziell auf das attributbasierte Word Spotting ausgelegt und daher per Definition in der Lage, PHOCs für Eingaben zu approximieren. Daher müssen keine besonderen Anpassungen vorgenommen werden, wie es beim LeNet und ResNet der Fall ist. Lediglich die Spatial Pyramid Pooling Schicht, die Teil des PHOCNets ist, um Eingaben belieber Größe verarbeiten zu können, wird durch eine Temporal Pyramid Pooling Schicht ersetzt. Folglich wird das CNN nun als TPP-PHOCNet bezeichnet. Sowohl die Schichten des Konvolutionsteils als auch die Schichten des MLPs bleiben unverändert. Tabelle 4.1.1 (mittlere Spalte) führt sämtliche Schichten des TPP-PHOCNets auf.

PHOCResNet

Die komplexeste der drei Netzarchitekturen, das PHOCResNet [Rus+18b], basiert auf dem ResNet50 [He+16]. Es handelt sich dabei um ein sogenanntes Residual Network, das gegen das Problem des Verfalls der Trainings-Accuracy tiefer Netze mit Hilfe von

sogenannten Residual Blocks vorgeht [He+16]. Die Grundidee lautet, die Schichten eines Blocks ein Residual Mapping lernen zu lassen, das sich als $H(x) = F(x) + x$ beschreiben lässt. Das x beschreibt die Eingabe eines Blocks und wird außerdem über eine Shortcut Connection weitergeleitet. $F(x)$ steht für die Ausgabe eines Blocks von Schichten. Die Summe von x und $F(x)$ ergibt $H(x)$. Das ResNet50 besteht aus speziellen Residual Blocks, den Residual Bottleneck Blocks [He+16], die aus aufeinanderfolgenen 1×1, 3×3 und 1×1 Faltungen aufgebaut sind. Die erste 1×1 Konvolutionsschicht reduziert die Dimensionalität der Eingabe, während die letzte 1×1 Konvolutionsschicht die ursprüngliche Dimension wiederherstellt [Rus+18b]. Abbildung 4.1.1 zeigt ein Beispiel eines Residual Bottleneck Blocks.

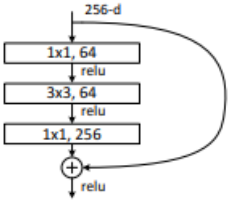


Abbildung 4.1.1: Beispiel für einen Bottleneck Building Block des ResNet50. Die Ausgabe der Konvolutionsschichten des Blocks werden mit der Identität der Eingabe (Shortcut Connection) verrechnet. Die Abbildung ist aus [He+16] entnommen.

Der Resnet50-Architektur entsprechend besteht der Konvolutionsteil des PHOCResNets aus einer 7×7 Konvolutionsschicht und einer 3×3 Max Pooling Schicht, auf die 16 Residual Bottleneck Blocks folgen. Insgesamt besteht der Konvolutionsteil des PHOCResNets aus 49 Konvolutionsschichten. Das Hinzufügen einer Temporal Pyramid Pooling Schicht ermöglicht das Bearbeiten von Eingaben beliebiger Größe. Das nachfolgende MLP entspricht dem MLP des TPP-PHOCNets, denn die einzelne vollvernetzte Schicht des ResNet50 liefert keine zufriedenstellenden Ergebnisse [Rus+18b]. Zwei verdeckte, vollvernetzte Schichten mit je 4096 Neruonen und eine Schicht, deren Neuronenanzahl der Attributanzahl entspricht und auf die eine Sigmoid-Schicht folgt, bilden das MLP zur Attributapproximation. Zuletzt werden im PHOCResNet die Strided Convolutions nach den Blöcken 7 und 13 entfernt, die nichts anderes als eine Form von Downsampling darstellen [Rus+18b]. Tabelle 4.1.1 (rechte Spalte) gibt eine Übersicht der im PHOCResNet enthaltenen Schichten.

4.2 DAPHOCNET

In diesem Kapitel wird das DAPHOCNet vorgestellt. Es handelt sich um eine Architektur, die eine erweiterte Version des TPP-PHOCNets darstellt und das Ziel hat, einer Verschlechterung der Leistungsfähigkeit eines Modells entgegenzuwirken, sobald es zu einem Domänenwechsel der Eingabedaten kommt. In diesem Kontext ist damit der Wechsel von synthetischen Trainingsdaten auf reale Testdaten gemeint. Die Konstruktion und das Training des DAPHOCNets orientieren sich dabei an den Arbeiten von *Kang et al.* [Kan+20] sowie *Ganin und Lempitsky* [GL15]. Die Architektur wird in Kapitel 4.2.1 anhand einer Visualisierung erläutert. Kapitel 4.2.2 befasst sich mit dem Diskriminator, um welchen das TPP-PHOCNet erweitert wird, damit Domain Adaptation möglich ist. Anschließend werden in Kapitel 4.2.3 wichtige Aspekte des Trainings des DAPHOCNets beschrieben.

4.2.1 *Aufbau*

Das DAPHOCNet setzt sich aus fünf essentiellen Grundbausteinen zusammen, die in Abbildung 4.2.1 visualisiert sind. Analog zum TPP-PHOCNet, das von der VGG16-Architektur [SZ14] inspiriert ist, werden in den Konvolutionsschichten 3×3-Filter mit Stride 1 verwendet. Auf alle Konvolutionsschichten folgen ReLUs, die für Nichtlinearität im Netz sorgen und zudem das Vanishing Gradient Problem verhindern. Die Anzahl der Filter pro Schicht nimmt in den tieferen Schichten zu, was zur Folge hat, dass in den ersten Schichten wenige Merkmale mit geringer Komplexität und in den tieferen Schichten mehr hochkomplexe Merkmale gelernt werden [SZ14]. Die beiden Pooling Schichten im vorderen Konvolutionsteil arbeiten mit 2×2-Maxpooling und Stride 2. Im Anschluss an den Konvolutionsteil des DAPHOCNets folgt eine 3-Level Temporal Pyramid Pooling Schicht (siehe Kapitel 2.3.2), die die Verarbeitung von Wortabbildern beliebiger Größe ermöglicht, ohne dass diese skaliert oder zurechtgeschnitten werden müssen. TPP bietet gegenüber klassischem Spatial Pyramid Pooling im Kontext von Word Spotting den Vorteil, dass die Schreibrichtung in die Berechnung mit einbezogen wird [SF18].

Nach dem TPP-Layer teilt sich die Architektur des DAPHOCNets in zwei Komponenten auf. Zum einen gibt es analog zum (TPP)-PHOCNet das MLP, das PHOCs approximiert. Es besteht aus drei vollvernetzten Schichten, wovon die ersten beiden jeweils aus 4096 Neuronen bestehen, auf die ReLUs folgen. Zusätzlich wird hier wie in [SF16] ein Dropout von 0.5 zur Regularisierung verwendet. Bei Regularisierung handelt es sich um ein Verfahren, das Overfitting in Netzwerken mit großer Parameteranzahl verhindern soll [SF18]. Dabei werden zufällig manche Aktivierungen eines Layers auf Null gesetzt. Die Anzahl der Neuronen in der letzten vollvernetzten

Schicht entspricht der Attributanzahl. Das MLP wird durch eine Sigmoid-Schicht abgeschlossen.

Die andere Komponente des DAPHOCNets, die auf den TPP-Layer folgt, ist ein einfacher Binärklassifikator bestehend aus einer vollvernetzten Schicht und einem einzelnen Neuron als Ausgabeschicht. Er wird im Folgenden auch als Diskriminator bezeichnet. Dem Diskriminator ist ein Gradient Reversal Layer vorgelagert.

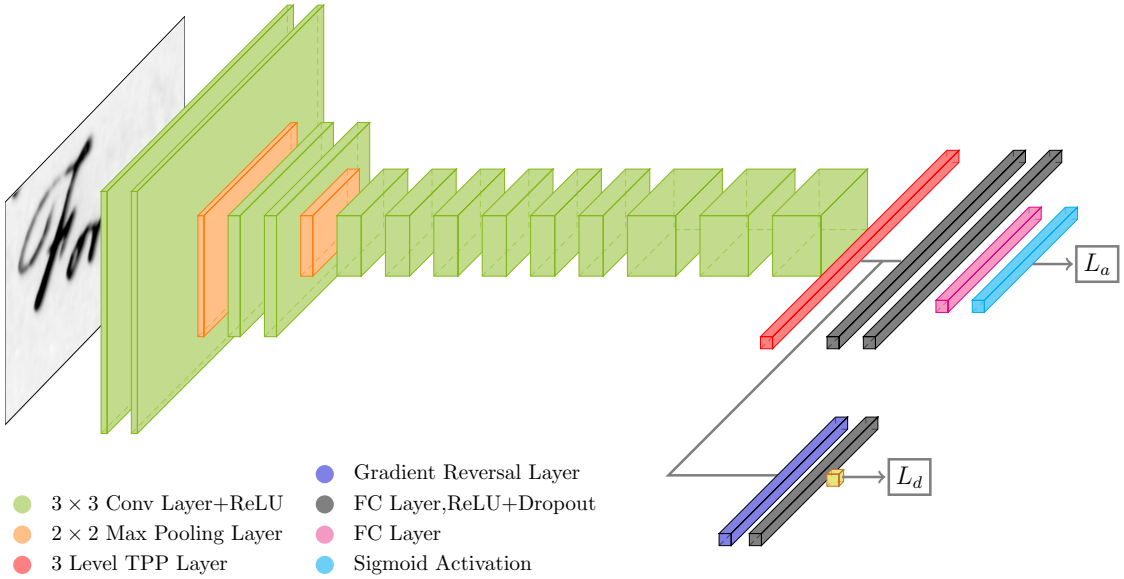


Abbildung 4.2.1: Die Abbildung zeigt die Architektur des DAPHOCNets. Die grünen Konvolutionsschichten, gefolgt von ReLUs, bilden zusammen mit zwei orange gefärbten Max Pooling Schichten den Konvolutionsteil des DAPHOCNets. Exakt wie im TPP-PHOCNet folgt ein Temporal Pyramid Pooling Layer (rot markiert). Darauf folgt die Aufteilung des DAPHOCNets in zwei Komponenten. Der für die Approximation von Attributvektoren zuständige Teil besteht aus drei vollvernetzten Schichten (FC), wobei die Neuronenanzahl der letzten der drei Schichten, visualisiert durch die magentafarbene Box, der Attributanzahl entspricht. Darauf folgt die Sigmoid-Aktivierung, hier cyan gefärbt. Die andere Komponente des DAPHOCNets ist ein Diskriminator. Er besteht aus einer vollvernetzten Schicht, auf die ein einzelnes Neuron als Ausgabeschicht folgt. Die dunkelblaue Box zeigt den vorgelagerten Gradient Reversal Layer (GRL).

4.2.2 *Binärklassifikation*

Domain Adaptation wird mit Hilfe eines Diskriminators umgesetzt, der lernt, Repräsentationen synthetischer und realer Eingabewortabbilder zu unterscheiden. Er besteht aus einer vollvernetzten Schicht und einem nachgelagerten Neuron zur binären Klassifikation. Die eingesetzte Kostenfunktion ist der BCE-Loss.
Die Grundidee lautet, den nach der Klassifikation berechneten Fehler mit umgekehrtem Vorzeichen an den Konvolutionsteil des Netzwerks zurückzupropagieren. Um den Fehler in das Netzwerk zurückzuführen und schließlich den Gradienten der Kostenfunktion bezüglich der Parameter zu berechnen, der zum Update der Parameter gebraucht wird, wird der Backpropagation Algorithmus verwendet (siehe Kapitel 2.2.4). Wichtig ist dabei, dass das Vorzeichen des Fehlers erst gewechselt wird, nachdem er durch den Diskriminator zurückpropagiert wurde. Dadurch wird ein gegensätzlicher Trainingseffekt im Netzwerk erzielt, der es durch das Lernen domäneninvarianter Merkmale im Konvolutionsteil schwieriger für den Diskriminator macht, diese zu unterscheiden [Kan+20][GL15]. Die Umkehrung des Vorzeichens wird durch eine dem Diskriminator vorgelagerte Schicht umgesetzt, die beim Forwardpass die Identitätsfunktion berechnet und beim Backwardpass das Vorzeichen des Fehlers umdreht. Diese Schicht wurde in Kapitel 4.2.1 als Gradient Reversal Layer eingeführt.

4.2.3 *Training*

Zur Umsetzung des Ziels, die synthetische Merkmalsdomäne an eine reale Merkmalsdomäne anzupassen, damit eine gute Leistungsfähigkeit des Netzes erreicht wird, muss der Konvolutionsteil des DAPHOCNets domäneninvariante Merkmale lernen. Die verwendeten synthetischen und realen Daten definieren dabei die Domänen, die es aneinander anzugleichen gilt.

Das Training des DAPHOCNets gliedert sich den beiden Domänen der Eingaben entsprechend in zwei Bereiche. Trainingsbeispiele der synthetischen Domäne tragen sowohl zum Optimierungsprozess des Approximatorteils als auch zum Optimierungsprozess des Diskriminators bei. Sie werden durch beide Komponenten des Netzwerks propagiert und dementsprechend kommt es zur Berechnung zweier Fehler, L_a und L_d. Dazu wird im DAPHOCNet die BCE-Lossfunktion verwendet (siehe Kapitel 2.2.3). Im Gegensatz zu den synthetischen Daten liegen reale Daten ohne Annotationen vor. Sie werden nur durch den Konvolutionsteil zur Merkmalsextraktion und anschließend in anderer Repräsentation durch den Diskriminator propagiert. Dabei wird nur eine Fehlerfunktion berechnet, deren Fehler L_d mittels Backpropagation durch den Diskriminator und im Anschluss, nach dem Vorzeichenwechsel im Gradient Reversal Layer, durch den Konvolutionsteil zurückpropagiert wird. Auf Grundlage dieser beiden

Gradienten wird das DAPHOCNet im min-max Stil trainiert, was die Minimierung des Approximatorfehlers und die Maximierung des Diskriminatorfehlers umfasst.

Gleichung 4.2.1 definiert den auf Basis von L_a und L_d zu berechnenden Gesamtloss des DAPHOCNets, $L(\theta_e, \theta_a, \theta_d)$. Dies entspricht einer leicht modifizierten Variante der Lossformel aus der Arbeit von *Kang et al.* [Kan+20] (siehe Kapitel 3.2). Sie wird um den Skalierungsfaktor α erweitert, der den berechneten Diskriminatorloss L_d und den Approximatorloss L_a auf die gleiche Größenordnung bringen soll. Vorexperimente haben ergeben, dass eine solche Anpassung nötig ist, da das Training sonst schlechte Resultate liefert. Hier hat sich die Wahl von $\alpha = 0.015$ als sinnvoll erwiesen. Das DAPHOCNet hat im Gegensatz zur Architektur von *Kang et al.* nicht das Ziel, Erkennungsaufgaben zu lösen. Daher wird nicht von einem Recognizer Loss L_r gesprochen, sondern vom Approximator Loss L_a, da PHOCs im Kontext von Word Spotting approximiert werden sollen.

$$L(\theta_e, \theta_a, \theta_d) = \sum_{x_i \in D_s} L_a(G_a(G_e(x_i)), y_i) - \alpha \lambda \sum_{x_j \in D_s \cup D_t} L_d(G_d(G_e(x_j)), d_j). \quad (4.2.1)$$

$G_e(x)$, $G_a(x)$ und $G_d(x)$ stehen für die Ausgaben des Konvolutionsteils (Merkmals**e**xtraktion), des vollvernetzten Teils (PHOC-**A**pproximation) und des **D**iskriminators. D_s und D_t bezeichnen Daten aus **S**ource- und **T**argetdomain (Quell- und Zieldomäne). Das λ ist ein Faktor, mit dem sich der Einfluss des Diskriminators auf den Konvolutionsteil regulieren lässt. Das Optimierungsziel lautet, ähnlich wie in [Kan+20], einen Sattelpunkt zu finden, sodass

$$\hat{\theta}_e, \hat{\theta}_a, = \underset{\theta_e, \theta_a}{\operatorname{argmin}} L(\theta_e, \theta_a, \theta_d) \quad (4.2.2)$$

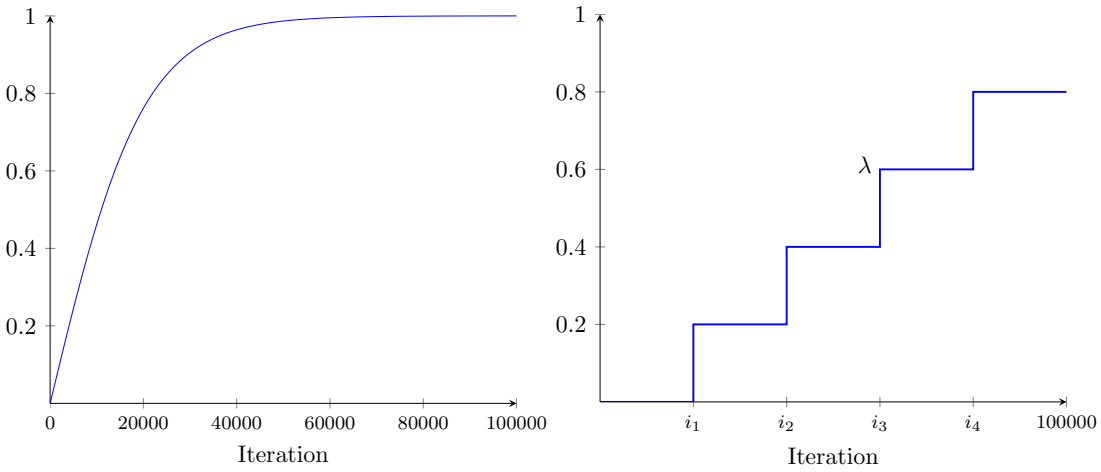
$$\hat{\theta}_d = \underset{\theta_d}{\operatorname{argmax}} L(\theta_e, \theta_a, \theta_d). \quad (4.2.3)$$

Im Folgenden werden zwei Varianten vorgestellt, auf die sich die Wahl des Wertes für λ in der Auswertung bezieht. In beiden Varianten gilt $\lambda \in [0, 1]$. Intuitiv kann λ daher als prozentualer Einfluss von L_d auf den Konvolutionsteil des Netzwerks betrachtet werden. Trivialerweise liefert die Wahl von $\lambda = 0$ Resultate, die denen eines auf synthetischen Daten trainierten TPP-PHOCNets ohne Domain Adaptation entsprechen. Die erste Variante, λ zu wählen und während des Trainings anzupassen, basiert auf der Arbeit von *Ganin und Lempitsky* [GL15], die ein mit dem Trainingsverlauf progressiv ansteigendes λ für ihre Architektur nutzen. Gleichung 4.2.4 definiert das progressive λ_{prog}. Die Variable p ergibt sich als Quotient aus der maximalen Iterationsanzahl und

der gegenwärtigen Iteration. Abbildung 4.2.2a visualisiert λ im Verlauf von 100000 Iterationen.

$$\lambda_{prog} = \frac{2}{1 + \exp^{-10 * \frac{1}{p}}} - 1 \tag{4.2.4}$$

Wird λ_{prog} für das Training gewählt, so nimmt der Diskriminator von Anfang an Einfluss auf das Training des Konvolutionsteils.



(a) Progressive λ

(b) Scheduled-λ

Abbildung 4.2.2: Abbildung 4.2.2a stellt die Entwicklung des progressiven λ (siehe Formel 4.2.4) über 100000 Iterationen dar. Abbildung 4.2.2b zeigt die Entwicklung des Schedule-λ über 100000 Iterationen. Hier geben i_1 bis i_4 die Iterationen an, die einer Schwelle entsprechen, an der λ erhöht wird.

Die zweite Variante λ zu wählen beruht auf der Beobachtung aus Vorexperimenten, dass es sinnvoll ist, den PHOC-Approximator für einige Iterationen ohne Einfluss des Diskriminators zu trainieren. Das hat auch zur Folge, dass der Diskriminator eine Zeit lang die Unterscheidung der Domänen lernen kann, ohne durch Merkmale, die tendenziell domäneninvarianter werden, in die Irre geführt zu werden. Die Idee lautet daher für diese Variante, λ nach und nach, bei gewissen Iterationsmeilensteinen in Schritten von 0.2 zu erhöhen, nachdem das Netz eine Weile mit $\lambda = 0$ trainiert hat. Abbildung 4.2.2b visualisiert diese Variante.

Neben weiteren Hyperparametern spielt für das Training des DAPHOCNets der Gewichtungsfaktor λ eine zentrale Rolle, der zur gegenseitigen Abstimmung der im Approximator- und Diskriminatorteil berechneten Losses L_a und L_d genutzt wird. Dementsprechend liegt der Fokus der Auswertung auf den vorgestellten Varianten, das λ für das Training zu wählen.

5 EXPERIMENTE UND EVALUATION

Im Folgenden werden Experimente zu den Methodiken aus Kapitel 4 beschrieben und ausgewertet. Kapitel 5.1 gibt eine Übersicht zu den verwendeten Datensätzen. Das Evaluierungsprotokoll wird in Kapitel 5.2 genauer beschrieben und Informationen zum Training Setup befinden sich in Kapitel 5.3.

In Kapitel 5.4 soll zunächst die Forschungsfrage beantwortet werden, zu welchen Leistungsverlusten die Verwendung synthetischer Trainingsdaten bei den in der Methodik vorgestellten Netzarchitekturen führt. Nachfolgend wird in Kapitel 5.5 untersucht, ob ein Diskriminator synthetische und reale Daten korrekt klassifizieren kann und wie sich die Neuronenanzahl des Diskriminators auf die Klassifizierungsleistung auswirkt. Als letztes wird in Kapitel 5.6 anhand des DAPHOCNets ausgewertet, ob und wie gut Domain Adaption Leistungsverluste bei einem Domänenwechsel von synthetischen Trainingsdaten auf reale Testdaten verringern kann. Dabei wird in den zugehörigen Experimenten vorallem der Einfluss des Gewichtungsfaktors λ untersucht. Am Ende wird in einem Vergleich mit der Literatur ausgewertet, wie hoch die Leistung des DAPHOCNets verglichen mit state-of-the-art Methoden ist.

5.1 DATENSÄTZE

Die Evaluierung der Methodik geschieht auf drei Benchmark-Datensätzen für segmentierungsbasiertes Word Spotting. Bei HW-Synth handelt es sich um einen synthetischen Datensatz. Die anderen beiden Datensätze, der George Washington Datensatz und der IAM Datensatz, umfassen sowohl Dokumente historischer Art, als auch moderne handschriftliche Dokumente.

5.1.1 *HW-Synth*

Der Mangel an Trainingswortabbildern mit echter Handschrift veranlasste *P. Krishnan und C. V. Jawahar* dazu, einen Datensatz mit synthetischen Wortabbildern zu erstellen, den HW-Synth Datensatz [KJ16]. Er enthält 1 Million synthetische Wortabbilder, die auf 750 verschiedenen Fonts basieren und natürlicher Handschrift besonders ähnlich sind. Aufgrund hoher Variabilität echter Handschrift werden im Synthetisierungsprozess stilbestimmende Parameter zufällig variiert, um diese natürliche Variabilität möglichst präzise nachzuahmen. Ein Teil des Hunspell Wörterbuchs diente als Quelle für 10000

einzigartige Wörter, von denen jedes Einzelne mit jeweils 100 zufällig ausgewählten Fonts zu entsprechenden Wortabbildern synthetisiert wurde [KJ16].

5.1.2 *George Washington*

Der George Washington Datensatz (GW) [LRM04] besteht aus 20 Seiten historischer, englischsprachiger Dokumente aus dem Jahr 1755. Die Dokumente wurden ursprünglich von George Washington und seinen Mitarbeitern verfasst. Da sie aber später von einem einzigen Schreiber kopiert wurden, ist der GW Datensatz ein Single Writer Datensatz [Gio+17]. Die Seiten liegen segmentiert und annotiert vor und umfassen 656 Textzeilen mit insgesamt 4894 Wörtern. Im Bereich der Auswertung und des Vergleichs verschiedener Word Spotting Methoden ist der GW Datensatz der meistverwendete Datensatz [Gio+17]. Die Experimente werden mit der üblicherweise genutzten Kreuzvalidierung [Alm+14] durchgeführt.

5.1.3 *IAM*

Der IAM Datensatz [MB02] umfasst 1539 Seiten modernen, englischen Texts. Es handelt sich dabei um handschriftlich verfasste Dokumente von 657 verschiedenen Autoren. Dadurch ergibt sich im Gegensatz zum GW Datensatz eine hohe Variabilität der Schreibstile. Insgesamt enthalten die Seiten 13353 Textzeilen, sowie 115320 Wörter. Alle Seiten sind vollständig segmentiert und es liegen Annotationen für alle Wörter vor.

5.2 EVALUIERUNGSPROTOKOLL

Die Auswertung folgt dem Protokoll aus [Alm+14], das sich zum Standard im Word Spotting Bereich entwickelt hat [SF18][KJ19][Gio+17]. Die Auswertung aller vorgestellten Architekturen wird auf den Testsplits der Datensätze GW und IAM vorgenommen. Die einzige Ausnahme bildet hier die Verwendung des HW-Synth Testsplits zur Auswertung der auf HW-Synth trainierten Netzarchitekturen im Rahmen der In-Domain Experimente in Kapitel 5.4. Die Trainingssplits von GW und IAM werden nur für In-Domain Experimente mit Annotationen verwendet. Für die Experimente, die sich auf die Untersuchung des Leistungsverlusts der Netzarchitekturen bei Verwendung synthetischer Trainingsdaten beziehen, wird der Trainingssplit des HW-Synth Datensatzes genutzt und die GW und IAM Trainingssplits werden ignoriert. Testsplits von GW und IAM stellen dabei die realen Testdaten. Das Training in den Diskriminatorexperimenten in Kapitel 5.5 basiert auf der Nutzung der GW und IAM Trainingssplits, wobei die vorliegenden Annotationen ignoriert werden. Im Fall der DAPHOCNet-Experimente in Kapitel 5.6 werden zusätzlich zum HW-Synth Trainingssplit die Trainingssplits von

GW bzw. IAM für das Training verwendet, jedoch ohne Beachtung der Annotationen. Analog zur Auswertung der anderen Experimente liefern die Testsplits von GW bzw. IAM die realen Testdaten.

Für QbE wird jedes Bild als Anfrage verwendet und für QbS jeder einzigartige String. Die Hyperparameterisierung entspricht weitestgehend derjenigen aus [SF18]. Ebenso folgt diese Arbeit den in [SF18] vorgestellten Augmentierungsansätzen.

Zur Messung der Leistungsfähigkeit wird in den Experimenten die mean Average Precision (mAP) verwendet. Die mAP gibt an, wie gut eine Liste von Antworten, die sogenannte Retrieval List, im Bezug auf die Relevanz der Antworten zur Anfrage sortiert ist. Intuitiv ist die mAP ein Maß dafür, wie weit vorne relevante Antworten in der Rückgabeliste stehen. Gleichung 5.2.1 definiert die mAP:

$$mAP = \frac{\sum_{q=1}^{Q} AP_q}{Q} \tag{5.2.1}$$

Die Variable Q steht für die Anzahl der Anfragen. Die mAP ergibt sich aus dem Mittelwert der Summe der durchschnittlichen Präzisionswerte aller Anfragen q. Gleichung 5.2.2 definiert die sogenannte Average Precision [SF18]

$$AP_q = \frac{\sum_{i=1}^{n} P_q(i) r_q(i)}{\text{number of relevant elements}} \tag{5.2.2}$$

für jede Anfrage q, wobei $P_q(i)$ für die Precision bei Betrachtung der Retrieval List für Anfrage q bis zum Index i steht:

$$P = \frac{\text{number of relevant retrieved elements}}{\text{number of retrieved elements}} \tag{5.2.3}$$

Der Funktionswert von $r_q(i)$ ist 1, wenn das i-te Element der Rückgabeliste im Bezug auf q relevant ist, und ansonsten 0 [SF18].

5.3 TRAINING SETUP

Für das Training aller vorgestellten Architekturen werden die BCE-Fehlerfunktion und Adaptive Moment Estimation (Adam) [KB17] mit einem Momentum von 0.9, einem Weight Decay von $5 \cdot 10^{-5}$ und einer initialen Lernrate von 10^{-4} verwendet. Es werden für die Experimente zum ersten Methodikteil Minibatches der Größe 10 genutzt. Die Anzahl der Trainingsiterationen beträgt in diesem Fall 100000. Dabei wird die Lernrate

nach 60000 Iterationen durch 10 dividiert. Die Diskriminator- und DAPHOCNet-Experimente werden mit Minibatches der Größe 20 durchgeführt, was jeweils 10 synthetischen und 10 realen Wortabbildern entspricht. Die initiale Lernrate von 10^{-4} wird in den diesen Experimenten nicht verändert. Der Skalierungsfaktor α (siehe Kapitel 4.2.3) bleibt unverändert auf 0.015. Als Labels dienen PHOC-Attributvektoren mit Leveleinteilung [2,3,4,5].

5.4 NETZARCHITEKTUREN

In diesem Kapitel werden PHOCLeNet, TPP-PHOCNet und PHOCResNet im Bezug auf Unterschiede im Leistungsverlust bei einem Domänenwechsel ausgewertet. Zunächst werden sie aber auf ihre In-Domain Leistungsfähigkeit überprüft, um die korrekte Funktionalität der Implementierung der Architekturen sicherzustellen. Dazu werden Resultate aus der Arbeit von *Rusakov et al.* zum Vergleich herangezogen.

In-Domain bedeutet hier, dass die Modelle auf dem Datensatz ausgewertet werden, auf dem sie trainiert wurden. Es gibt lediglich in den Mengen der ausgewählten Trainings- und Testbeispiele Unterschiede. Tabelle 5.4.1 gibt die Übersicht der Resultate.

Domäne	GW		IAM	
Architektur	QbE	QbS	QbE	QbS
PHOCLeNet	71.03	75.78	10.26	16.54
PHOCLeNet [Rus+18b]	69.78	80.68	13.55	27.86
TPP-PHOCNet	97.23	95.33	82.21	91.79
TPP-PHOCNet [Rus+18b]	**97.29**	**98.47**	85.03	93.22
PHOCResNet	84.83	68.25	83.28	91.38
PHOCResNet [Rus+18b]	95.90	97.91	**88.35**	**94.69**

Tabelle 5.4.1: Resultate für die In-Domain Experimente nach 100000 Iterationen in mAP [%] mit Vergleichsresultaten aus [Rus+18b].

Es zeigt sich, dass vergleichbare Werte für das PHOCLeNet und das TPP-PHOCNet vorliegen, wobei das PHOCLeNet wie erwartet aufgrund geringer Filteranzahl die schlechtesten Leistungen zeigt. Einzig das PHOCResNet liefert auf GW nicht die Leistung, die zum Vergleichswert aus [Rus+18b] passt. Da es sich um die komplexeste der drei Architekturen handelt, ist es möglich, dass sich das schlechtere GW Ergebnis auf Unterschiede in den verwendeten Frameworks zurückführen lässt. Die Resultate auf IAM sind jedoch nicht im gleichen Maß schlechter, sondern geringfügiger.

Nachfolgend geht es nun um die Untersuchung von Leistungsverlusten bei einem Domänenwechsel von synthetischen Trainingsdaten auf reale Testdaten. Es wird für die drei genannten Netzarchitekturen ausgewertet, wie hoch ihre Leistungsfähigkeit nach dem Training bei einem Domänenwechsel auf realen Testdaten ist. Die Motivation hierfür liegt in der Einsparung des kostspieligen Annotationsaufwands realer Trainingsdaten. Tabelle 5.4.2 zeigt die QbE und QbS Ergebnisse für die drei Netze, die auf dem synthetischen Datensatz HW-Synth für 100000 Iterationen trainiert wurden.

Testset	HW-Synth		GW		IAM	
Architektur	QbE	QbS	QbE	QbS	QbE	QbS
PHOCLeNet	8.97	20.87	9.96	3.90	2.79	4.97
TPP-PHOCNet	**98.30**	**98.90**	**47.04**	**53.44**	16.30	39.95
PHOCResNet	96.57	97.45	39.94	52.74	**25.14**	**49.91**

Tabelle 5.4.2: Resultate für die QbE und QbS Experimente mAP [%] zu den Netzarchitekturen, die auf HW-Synth trainiert wurden.

Es ist erkennbar, dass die Leistung aller Modelle im Vergleich zur In-Domain Auswertung bei einem Domänenwechsel deutlich sinkt. Analog zu den In-Domain Experimenten weist das PHOCLeNet auch hier schwache Resultate auf allen Datensätzen auf, die sich bis auf die QbS-mAP auf HW-Synth unter einem Wert von 10% befinden. Die besten Leistungen auf GW zeigt das TPP-PHOCNet mit 47.04 für QbE und 53.44 für QbS. Für IAM liefert das PHOCResNet mit 25.14 respektive 49.91 die besten Ergebnisse dieser Experimente. Dennoch ist hier die Leistung im Vergleich zu den In-Domain Ergebnissen auf IAM, die bei 83.28 und 91.38 liegen, deutlich niedriger. Insgesamt kann gefolgert werden, dass die Architekturen PHOC-Vektoren für reale Daten zwar bis zu einem gewissen Grad approximieren können, dabei aber keine Resultate erreichen, die auf Ebene der In-Domain Ergebnisse liegen. In den folgenden Experimenten wird daher die bekannte Standardarchitektur, das TPP-PHOCNet, in Kombination mit Domain Adaptation darauf untersucht, ob sich die hier gezeigten Ergebnisse verbessern lassen. Die in Tabelle 5.4.2 aufgeführten Resultate bilden dafür die Baseline.

5.5 DISKRIMINATOR

Zur Durchführung von Domain Adaptation benötigt ein CNN (hier das TPP-PHOCNet) einen Diskriminator, der zur bestehenden Architektur hinzugefügt wird wie in Kapitel 4.2.1 gezeigt. Im Folgenden wird untersucht, ob ein Diskriminator überhaupt in der Lage ist, das Klassifikationsproblem auf synthetischen und realen Daten zu lösen. Dabei wird auch untersucht, wie sich drei verschiedene Anzahlen von Neuronen im Diskriminator auf die Klassifizierungsleistung auswirken. Es wurden über 10000 Iterationen Minibatches von je 10 synthetischen und 10 realen Wortabbildern pro Iteration durch den Diskriminator propagiert, worauf jeweils ein Optimierungsschritt folgte. Bei den verschiedenen Neuronenanzahlen handelt es sich um 128, 512 und 1152, wobei im Fall von 1152 Neuronen zwei verdeckte Schichten mit 1024 und 128 Neuronen verwendet wurden statt einer einzelnen verdeckten Schicht. Tabelle 5.5.1 bildet die Ergebnisse der Untersuchung ab.

	GW		IAM	
Neuronen	HW Acc	GW Acc	HW Acc	IAM Acc
1152	100	99.84	99.97	99.14
512	99.34	100	99.79	99.90
128	99.51	100	98.43	100

Tabelle 5.5.1: Resultate für die Diskriminatorexperimente nach 10000 Iterationen in Genauigkeit [%]

Für beide verwendeten realen Datensätze hat der Diskriminator Ergebnisse im Bereich von 98 bis 100% Genauigkeit erzielt. Die Genauigkeit entspricht dem Quotienten aus korrekt klassifizierten Beispielen und der Gesamtanzahl der Beispiele. Der Diskriminator war unabhängig von der getesteten Neuronenanzahl in der Lage, mit nahezu 100% Genauigkeit die Domäne der Eingabe zu schätzen. Auf Basis dieser Ergebnisse wird in den DAPHOCNet-Experimenten ein Diskriminator mit zunächst 128 Neuronen verwendet, um die Architektur so einfach wie möglich zu halten. Zum Vergleich der Diskriminator-Neuronenanzahl im Kontext der Leistungsfähigkeit des DAPHOCNets wird zusätzlich ein ausgewähltes DAPHOCNet-Experiment mit 512 Neuronen im Diskriminatorteil durchgeführt, siehe Kapitel 5.6.

5.6 DAPHOCNET

In diesem Kapitel werden Experimente zum zweiten Teil der Methodik erläutert und ausgewertet, um herauszufinden, ob einer Verschlechterung der Leistungsfähigkeit bei einem Domänenwechsel mit Hilfe von Domain Adaptation entgegengewirkt werden kann und wie weit sich damit der Abstand der in Kapitel 5.4 diskutierten Baselineresultate zu In-Domain Resultaten reduzieren lässt. Die Leistungsfähigkeit des DAPHOCNets wird in zwei verschiedenen Experimentvarianten untersucht und später mit der Literatur verglichen. Die Varianten beziehen sich auf den in Kapitel 4.2.3 eingeführten Gewichtungsfaktor λ, der für das Training des DAPHOCNets eine zentrale Rolle spielt. Er steuert, wie stark sich der Diskriminatorfehler innerhalb des Optimierungsprozesses auf den Konvolutionsteil des DAPHOCNets auswirkt. Er bestimmt somit maßgeblich mit, welche Merkmale gelernt werden.

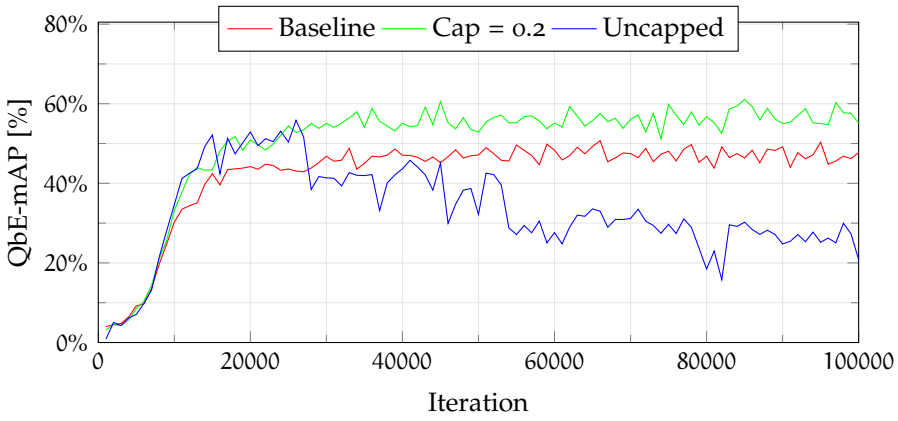
5.6.1 *Progressive* λ

In der ersten Variante wird untersucht, wie sich ein progressiver Gewichtungsfaktor λ_{prog} auf die Leistungen des DAPHOCNets auswirkt. Dabei wird λ_{prog} in fünf Experimenten nach oben beschränkt. Jedes Experiment wird mit einer anderen λ_{prog}-Schranke durchgeführt. Die Ergebnisse werden mit der Baseline verglichen, die $\lambda_{prog} = 0$ entspricht, da der Diskriminator dann keinen Einfluss auf das Lernen von Merkmalen im Konvolutionsteil des DAPHOCNets hat. Tabelle 5.6.1 zeigt die Resultate der Experimente.

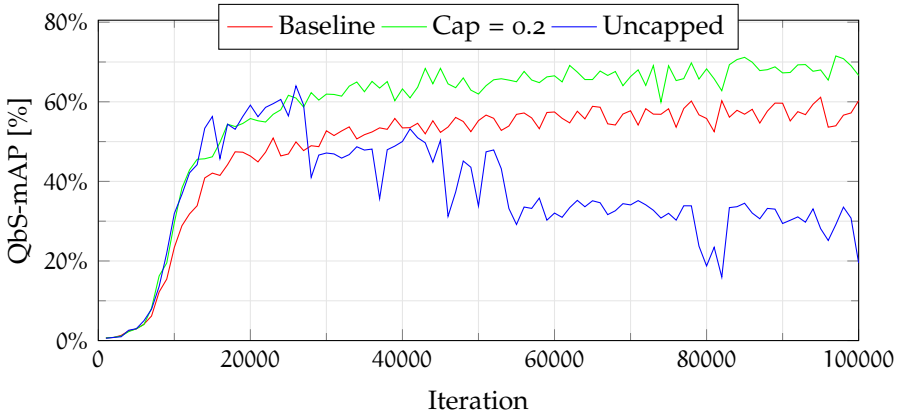
Bis auf den Fall, in dem der Gewichtungsfaktor in einem Experiment auf GW auf 1.0 beschränkt wurde, zeigt diese Experimentreihe Resultate auf, die in allen Fällen die Baseline verbessern. Die Beschränkung von λ_{prog} auf 1.0 wird im Folgenden auch als Uncapped oder unbeschränkt bezeichnet, da die Funktion (siehe Abbildung 4.2.2a) keinen größeren Wert als 1.0 annehmen kann. Auf dem GW Datensatz hat die Beschränkung von λ_{prog} auf maximal 0.6 mit 59.14 für QbE und 68.05 für QbS die besten Ergebnisse geliefert. Auf dem IAM Datensatz haben sich mit 26.55 für QbE und 53.59 für QbS die höchsten Resultate bei einer Begrenzung des Gewichtungsfaktors auf 1.0 gezeigt. Auf GW verschlechtert sich die Leistung bei Beschränkung von λ_{prog} auf 1.0. Dies ist gut am Verlauf der blauen Kurve in den Abbildungen 5.6.1a und 5.6.1b zu sehen. Wird der Gewichtungsfaktor beim Training auf GW nicht ausreichend beschränkt, verschlechtert sich die Leistung schon früh im Training bei ca. 27000 Iterationen. Die Beschränkung auf 0.2 zeigt hingegen konstant über der Baseline liegende Leistungen, wenn wie hier ein langer Trainingsverlauf über 100000 Iterationen betrachtet wird.

	GW		IAM	
λ-Schranke	QbE	QbS	QbE	QbS
0.0 (Baseline)	47.04	53.44	16.30	39.95
0.2	55.10	63.24	19.16	43.72
0.4	53.86	63.87	19.59	44.85
0.6	**59.14**	**68.05**	22.46	47.54
0.8	55.21	66.74	21.03	46.74
1.0	43.63	50.03	**26.55**	**53.59**

Tabelle 5.6.1: Resultate für die QbE und QbS Experimente in mAP [%] für das progressive λ_{prog} mit oberer Schranke. Die Modelle wurden bei Verwendung des GW Datensatzes für 40000 Iterationen und bei Verwendung des IAM Datensatzes für 90000 Iterationen trainiert.



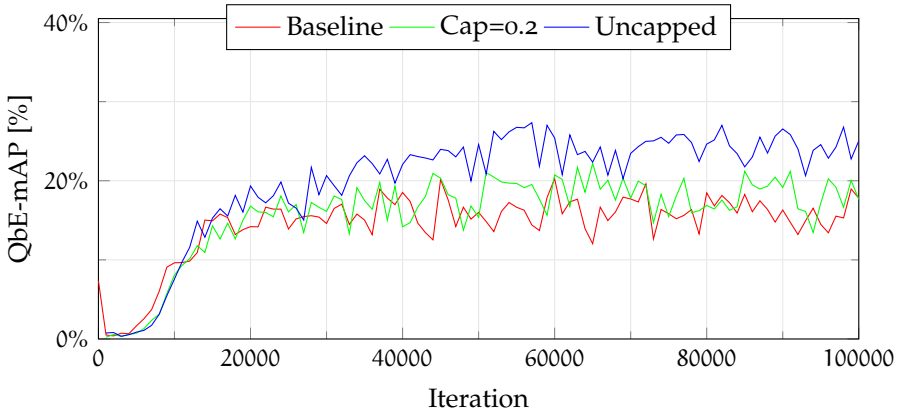
(a) Query-by-Example



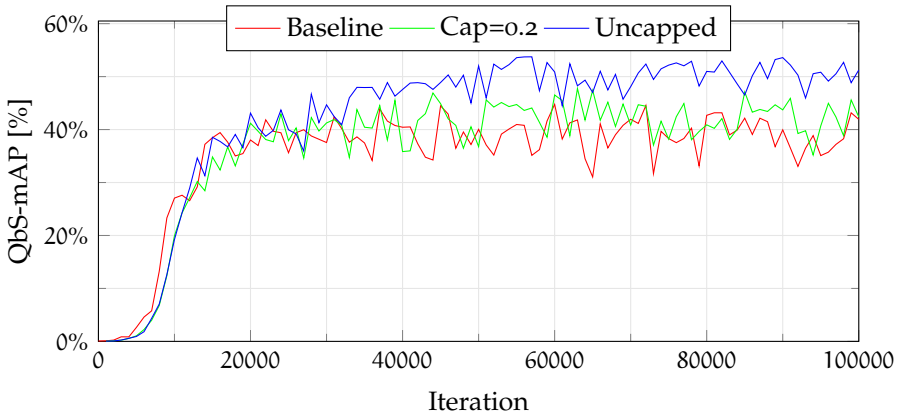
(b) Query-by-String

Abbildung 5.6.1: Visualisierung der Trainingsverläufe des DAPHOCNets für ausgewählte Experimente. Das Modell wurde auf dem GW Datensatz mit λ_{prog} über 100000 Iterationen trainiert.

Werden die Abbildungen 5.6.2a und 5.6.2b betrachtet, die sich auf die IAM Experimente beziehen, so zeigt sich für einen unbeschränkten Gewichtungsfaktor der beste Trainingsverlauf über 100000 Iterationen. Dies könnte ein Hinweis darauf sein, dass es beim deutlich kleineren GW Datensatz zu Overfitting kommt, sobald λ_{prog} einen gewissen Wert überschreitet. Bemerkenswert ist bei den IAM Experimenten, dass auch die λ_{prog}-Variante, die die kleinsten Verbesserungen bei maximal 90000 Iterationen (siehe Tabelle 5.6.1) gebracht hat, konvergierendes Verhalten zeigt, was im Gegensatz zum unbeschränkten GW Experiment steht.



(a) Query-by-Example



(b) Query-by-String

Abbildung 5.6.2: Visualisierung der Trainingsverläufe des DAPHOCNets für ausgewählte Experimente. Das Modell wurde jeweils auf dem IAM Datensatz mit λ_{prog} über 100000 Iterationen trainiert.

5.6.2 λ-*Scheduling*

Die zweite Variante durchgeführter DAPHOCNet-Experimente umfasst die Untersuchung verschiedener Schedulings für den Gewichtungsfaktor λ im Bezug auf die Anzahl bereits durchlaufener Iterationen. Vorexperimente haben gezeigt, dass es sinnvoll ist, das DAPHOCNet zunächst ohne Diskriminatoreinfluss zu trainieren. Daher wurden die Schedulings so eingestellt, dass nicht mit sofortigem Diskriminatoreinfluss trainiert wird, wie es in der ersten Experimentreihe mit progressivem λ_{prog} der Fall war. Tabelle 5.6.2 zeigt die verwendeten Schedulings, die alle bei den genannten Iterationsschwellen eine Erhöhung von λ um jeweils 0.2 vorsehen:

λ bei Iteration x	0.2	0.4	0.6	0.8
Schedule 1	5000	10000	15000	20000
Schedule 2	10000	15000	20000	25000
Schedule 3	10000	20000	30000	40000

Tabelle 5.6.2: Auflistung der Iterationszahlen bei denen λ jeweils um 0.2 höher gesetzt wird.

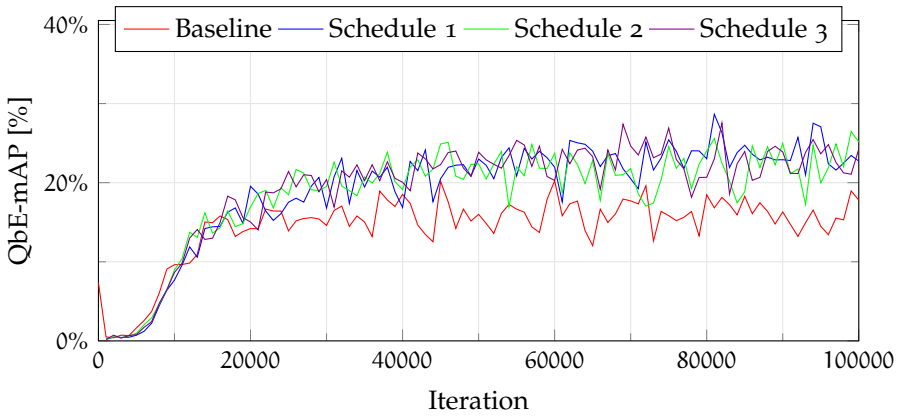
Die Mindestanzahl an Iterationen ohne Diskriminatoreinfluss beträgt nach Tabelle 5.6.2 5000 für Schedule 1. Tabelle 5.6.3 zeigt die Ergebnisse der Scheduling-Experimente.

	GW		IAM	
	QbE	QbS	QbE	QbS
Baseline	47.04	53.44	16.30	39.95
Schedule 1	56.09	64.19	**24.96**	**50.89**
Schedule 2	56.41	67.77	23.89	50.76
Schedule 3	**59.32**	**69.06**	22.90	48.55

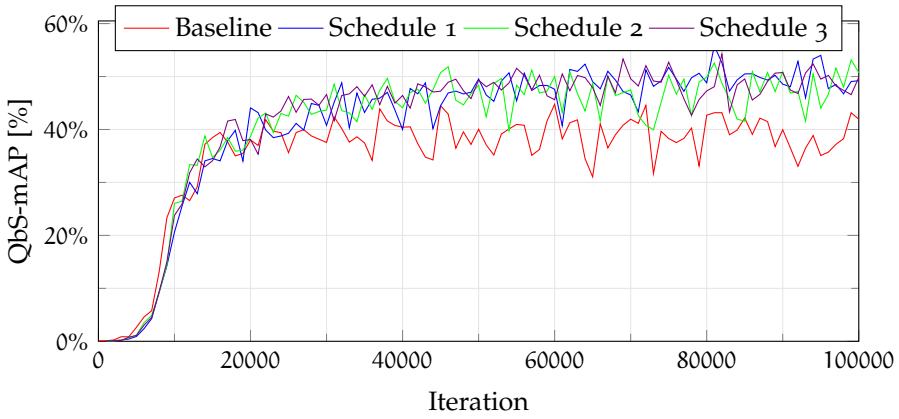
Tabelle 5.6.3: Resultate für die QbE und QbS Experimente in mAP [%] mit verschiedenen λ Schedulings. Die Modelle wurden bei Verwendung des GW Datensatzes für 40000 Iterationen und bei Verwendung des IAM Datensatzes für 90000 Iterationen trainiert.

Sowohl die Resultate auf dem GW Datensatz als auch die Resultate auf dem IAM Datensatz zeigen Verbesserungen im Vergleich zur Baseline. Auf GW bezogen zeigt sich das beste Ergebnis mit Schedule 3. Dort verbessert sich das Baselineresultat für QbE von 47.04 durch die Domain Adaptation auf 59.32 und für QbS von 53.44 auf

69.06. Die geringste Verbesserung, entstanden mit Schedule 1, liegt immerhin noch bei 9.05 Prozentpunkten für QbE und bei 11.11 Prozentpunkten für QbS. Das beste Resultat auf IAM konnte mit Schedule 1 erzielt werden, was darauf schließen lässt, dass hier relativ frühe, nicht zu weit auseinanderliegende λ-Erhöhungen positiv auf das Training wirken. Für Schedule 1 liegt die QbE-mAP bei 24.96, was im Vergleich zur Baseline von 16.30 eine gute Verbesserung darstellt. Die QbS-mAP von 50.89 ist ebenfalls höher als die bei 39.95 liegende Baseline. Da sich positive Resultate für alle Schedulings auf beiden Datensätzen zeigen, kann zumindest vorsichtig der Schluss gezogen werden, dass sich eine Erhöhung des Gewichtungsfaktors λ zu wenigen, bestimmten Zeitpunkten positiv auf die Leistung des DAPHOCNets auswirkt, nachdem es eine Zeit lang mit $\lambda = 0$ trainiert wurde.



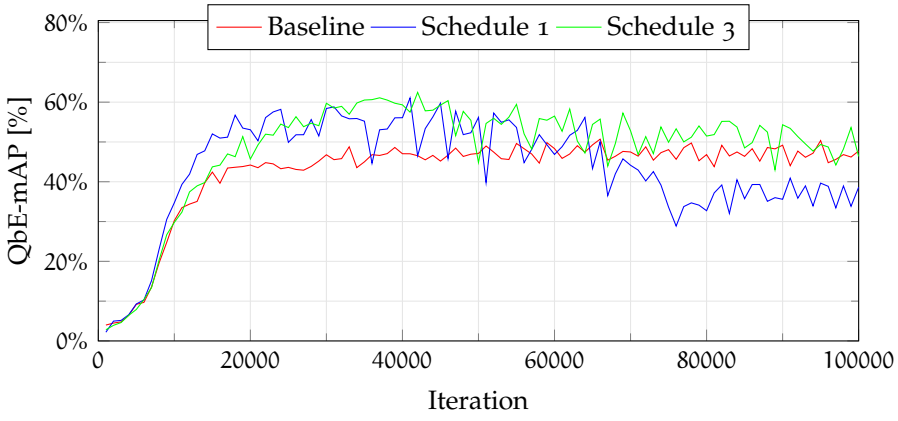
(a) Query-by-Example



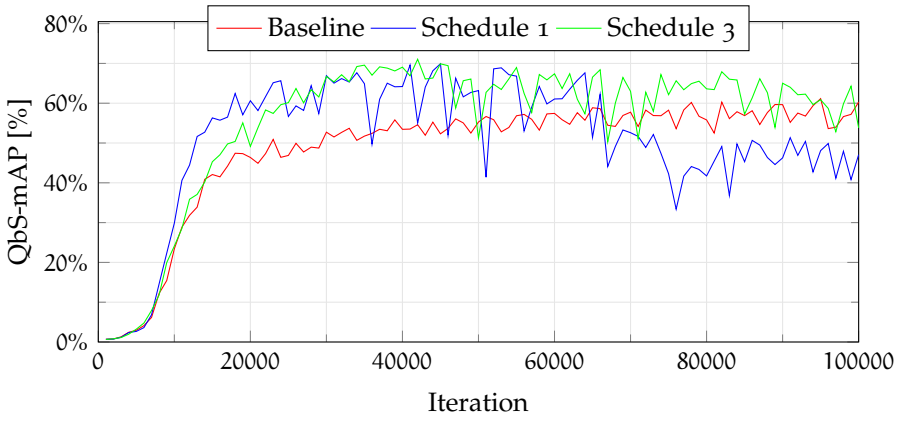
(b) Query-by-String

Abbildung 5.6.3: Visualisierung der verschiedenen Trainingsverläufe des DAPHOCNets. Alle dargestellten Experimente wurden jeweils auf dem IAM Datensatz mit λ-Scheduling und 100000 Iterationen durchgeführt.

Die Abbildungen 5.6.3a und 5.6.3b zeigen, wie die mAP-Werte der Scheduling-Experimente auf IAM über eine Trainingsdauer von 100000 Iterationen jeweils gegen Ende 5 bis 10 Prozentpunkte über der Baseline konvergieren, was die Schlussfolgerung stützt, dass Domain Adaptation mit bestimmten λ-Schedulings Leistungsverluste vermindert, die durch Nutzung synthetischer Trainingsdaten entstehen. Jedoch zeigen die Abbildungen 5.6.4a und 5.6.4b, dass dies nicht für alle Scheduling Varianten auf dem GW Datensatz gilt, wenn das Training die in Tabelle 5.6.3 vorgeschlagene Iterationsanzahl von 40000 übersteigt. Die Resultate für Scheduling 1 auf GW zeigen starke Leistungseinbrüche ab Iteration 65000.



(a) Query-by-Example



(b) Query-by-String

Abbildung 5.6.4: Visualisierung der verschiedenen Trainingsverläufe des DAPHOCNets für ausgewählte Expemirente. Alle Modelle wurden jeweils auf dem GW Datensatz mit λ-Scheduling für 100000 Iterationen trainiert.

Der beste Trainingsverlauf auf dem GW Datensatz über 100000 Iterationen, der mit Scheduling 3 erzielt wurde, zeigt, dass die mAP ab ca. 65000 Iterationen beginnt, sich der Baseline mAP anzunähern. Aus den Beobachtungen zum besten und schlechtesten Scheduling lässt sich für den GW Datensatz schließen, dass das DAPHOCNet zufriedenstellende Ergebnisse liefert, wenn die Anzahl der Trainingsiterationen unter 65000 bleibt. Generell scheinen relativ späte und weit auseinanderliegende Erhöhungen des Gewichtungsfaktors λ auf GW für gute Resultate zu sorgen.

5.6.3 *Diskriminatorgröße*

Die bisherigen Experimente haben zwar gezeigt, dass der Domain Adaptation Ansatz, der einen Diskriminator verwendet, die Baselineresultate verbessert, aber sie haben den Fokus auf den Gewichtungsfaktor λ und nicht auf die Komplexität des Diskriminators gelegt, die ausschlaggebend für die Leistung des DAPHOCNets sein könnte. Zuletzt wird daher untersucht, wie sich die Größe des Diskriminators auf die Resultate im Kontext eines ausgewählten Schedulings, hier Scheduling 3, auswirkt. Tabelle 5.6.4 zeigt die Resultate des Experiments.

	GW		IAM	
Neuronen - Experiment	QbE	QbS	QbE	QbS
128 - Baseline	47.04	53.44	16.30	39.95
128 - Schedule 3	59.32	69.06	22.90	48.55
512 - Schedule 3	**62.14**	**70.66**	**23.58**	**51.12**

Tabelle 5.6.4: Resultate für das Schedule 3 Experiment mit 128 und 512 Neuronen im Diskriminator in mAP [%]. Die Modelle wurden bei Verwendung des GW Datensatzes für 40000 Iterationen und bei Verwendung des IAM Datensatzes für 90000 Iterationen trainiert.

Die Verwendung von 512 Neuronen im Diskriminator hat die schon über der Baseline befindlichen Resultate noch verbessert, sowohl auf GW mit 62.14 für QbE und 70.66 für QbS als auch auf IAM mit 23.58 für QbE und 51.12 für QbS. Dies lässt die Schlussfolgerung zu, dass die Größe des Diskriminators trotz sehr ähnlichen Ergebnissen, was die Genauigkeit der Klassifikationen zu den getesten Diskriminatorgrößen angeht, einen merklichen Einfluss auf die Leistungsfähigkeit des DAPHOCNets hat. Es lässt sich jedoch nicht ohne weitere Experimente schließen, wo dabei Ober- bzw. Untergrenze liegen.

5.6.4 *Vergleich*

Die Experimente haben gezeigt, dass insbesondere auf GW feste Schedulings zur Festlegung des Gewichtungsfaktors λ dafür sorgen, dass mittels Domain Adaptation den Leistungsverschlechterungen eines CNNs bei Nutzung synthetischer Trainingsdaten entgegengewirkt werden kann. Die Variante, den Gewichtungsfaktor λ progressiv im Trainingsverlauf zu erhöhen, liefert auf IAM die besten Resultate. Auf GW ist die im Trainingsverlauf gemessene Leistung volatiler als bei der Scheduling Variante, aber sie übertrifft trotzdem die Baseline. Obwohl die Experimente in fast allen Fällen Verbesserungen gegenüber den Baselineresultaten aufweisen, erreicht die Leistungsfähigkeit des DAPHOCNets nicht das In-Domain Niveau, das zu Beginn diskutiert wurde.

Nachfolgend werden die besten DAPHOCNet-Resultate in Tabelle 5.6.5 mit der Literatur verglichen. Das angegebene DAPHOCNet-Resultat auf GW basiert auf der Verwendung des dritten λ-Schedules und einem Diskriminator mit 512 Neuronen (siehe Tabelle 5.6.4). Das DAPHOCNet-Resultat auf IAM folgt aus der Verwendung des unbeschränkten λ_{prog} (siehe Tabelle 5.6.1).

Testset	GW		IAM	
Methode	QbE	QbS	QbE	QbS
DAPHOCNet	62.14	70.66	26.55	53.59
Gurjar et al. [SGF17]	39.89	48.92	26.21	36.57
Wolf et al. [WF20]	46.60	57.90	16.00	39.50
Wolf et al. [WBF20]	**69.20**	**72.30**	**39.00**	**64.10**
Sudholt et al. [SF18]	97.90	*97.90*	85.50	93.40
Krishnan et al. [KJ19]	*98.20*	-	*92.40*	*94.00*

Tabelle 5.6.5: Vergleich mit der Literatur. Die Resultate sind in mAP [%] angegeben. Die besten Resultate unter Nutzung synthetischer Trainingsdaten sind fett gedruckt und die besten In-Domain Resultate sind kursiv dargestellt.

Die Leistung des DAPHOCNets übertrifft die der Methoden aus [SGF17] und [WF20], die auch auf dem PHOCNet und der Verwendung von HW-Synth als Trainingsdatensatz basieren. Sie unterliegt aber dem in [WBF20] präsentierten Ansatz, der seine Leistungsfähigkeit aus der Verwendung eines synthetischen Datensatzes bezieht, den die Autoren basierend auf Grid Augmentation und einem anderen Vokabular erstellt haben, das nicht dem für HW-Synth verwendeten Vokabular entspricht. Hier liegt

die Folgerung nahe, dass sich die Resultate aus [WBF20] mit Hilfe des präsentierten Domain Adaptation Ansatzes sogar noch verbessern lassen.

Die unteren zwei Tabellenzeilen zeigen als Vergleichswerte die Resultate von In-Domain Experimenten, die vollständig überwacht mit realen annotierten Trainingsdaten durchgeführt wurden. Die hohe Leistungsfähigkeit dieser Ansätze kommt jedoch zu hohen Kosten, da repräsentative Trainingsdaten manuell annotiert werden müssen. Es zeigt sich, dass das DAPHOCNet und insbesondere die Methode aus [WBF20] zwar den Leistungsabstand zu den In-Domain Ansätzen verringern, aber nicht einholen.

6 FAZIT

Convolutional Neural Networks haben sich durch ihre Leistungsfähigkeit in der Computervision insbesondere in der Analyse handschriftlicher Dokumente mittels Word Spotting etabliert. Das speziell auf Word Spotting ausgelegte PHOCNet gilt in diesem Kontext als eine state-of-the-art Methode, die sowohl QbE als auch QbS verarbeiten kann. Es bildet Wortabbilder auf Attributrepräsentationen ab, die einfache Distanzvergleiche im Attributraum ermöglichen. Das Problem bei der Verwendung solcher CNNs ist nach wie vor, dass Trainingsdaten kostspielig in der Erstellung sind. Synthetische Trainingsdaten basierend auf Computerfonts stellen eine günstige Alternative zu echten Trainingsdaten dar, weil manueller Annotationsaufwand entfällt. Ihre Verwendung ist jedoch mit Leistungsverlusten im Bezug auf eine Auswertung mit echten Zieldaten verbunden.

In dieser Arbeit wurde daher zunächst untersucht, wie hoch Leistungverluste bei einem Domänenwechsel von synthetischen Trainingsdaten auf reale Zieldaten ausfallen. Grundlage der Untersuchung bildeten dabei drei auf das Word Spotting zugeschnittene CNNs, das PHOCLeNet, das TPP-PHOCNet und das PHOCResNet. Die Experimente zeigten auf allen Architekturen große Leistungsverluste bei einem Domänenwechsel gegenüber In-Domain Experimenten. Nachfolgend wurde das in der Methodik vorgestellte DAPHOCNet, das einem auf Domain Adaptation auslegten TPP-PHOCNet entspricht, experimentell ausgewertet. Die Motivation lag hierbei in der Möglichkeit, der in den Baselineexperimenten gezeigten Verschlechterung der Leistung bei Einsatz synthetischer Trainingsdaten mittels Domain Adaptation entgegenzuwirken. Die Experimente haben gezeigt, dass eine Anpassung der synthetischen Merkmalsdomäne an eine neue Zieldomäne für Resultate sorgt, die die Baseline übertreffen. Dies gilt für beide in dieser Arbeit verwendeten Zieldatensätze, GW und IAM.

Der Fokus der DAPHOCNet-Experimente lag auf dem Gewichtungsfaktor λ, der zur Steuerung des Einflusses des Diskriminators auf den Konvolutionsteil des DAPHOCNets eingesetzt wird. In den Experimenten wurden dabei zwei Varianten zur Festlegung von λ während des Trainings ausgewertet. Bei den Varianten handelte es sich um eine progressive Erhöhung und eine stufenweise von der Iterationszahl abhängige Erhöhung. Die besten Resultate auf dem GW Datensatz haben sich bei einer stufenweisen Erhöhung von λ gezeigt. Für IAM lieferte die progressive Variante das beste Ergebnis. In beiden Fällen konnte die Baseline um ca. 10 Prozentpunkte verbessert werden, was insgesamt klar für die Verwendung von Domain Adaptation im Rahmen der Verwendung synthetischer Trainingsdaten spricht.

Im Kontext der hier vorgestellten Experimente könnte es sich als sinnvoll erweisen, den Einfluss der Komplexität des Diskriminators auf die Leistung des DAPHOCNets zu untersuchen, da in einem der Experimente eine Leistungsverbesserung durch eine Erhöhung der Neuronenanzahl des Diskriminators gezeigt werden konnte. Auch scheint eine Verbesserung der Resultate aus [WBF20] mittels Domain Adaptation möglich zu sein. In der Arbeit von *Wolf et al.* wird eine Methode für annotationsfreies Word Spotting basierend auf synthetischen Trainingsdaten vorgeschlagen, die es ermöglicht, den synthetischen Trainingsdatensatz während der Synthese an den Stil der Zieldomäne anzupassen, was eine höhere Leistung verspricht. Mit anschließendem Finetuning ergeben sich state-of-the-art Resultate für annotationsfreies Word Spotting. Zur weiteren Verbesserung der Resultate könnte Domain Adaptation während des Trainings des initialen Modells eingesetzt werden. Die Kombination aus einer Stilanpassung der synthetischen Trainingsdaten an den Stil des Zieldatensatzes , einer Anpassung der gelernten Merkmale im Training mittels des in dieser Thesis vorgestellten Domain Adaptation Ansatzes und schließlich nachfolgendem Finetuning eignet sich für vielversprechende zukünftige Untersuchungen.

LITERATUR

[AAKM16] Rashad Ahmed, Wasfi Al-Khatib und Sabri Mahmoud. „A Survey on handwritten documents word spotting“. In: *International Journal of Multimedia Information Retrieval* 6 (2016).

[Alm+14] J. Almazán, A. Gordo, A. Fornés und E. Valveny. „Word Spotting and Recognition with Embedded Attributes“. In: *TPAMI*. 2014.

[AFV13] J. Almazán, A. Fornés und E. Valveny. „Deformable HOG-Based Shape Descriptor“. In: *International Conference on Document Analysis and Recognition (IJDAR)*. 2013, S. 1022–1026.

[AD07] Esra Ataer und Pinar Duygulu. „Matching Ottoman Words: An Image Retrieval Approach to Historical Document Indexing“. In: *ACM International Conference on Image and Video Retrieval (CIVR)*. Amsterdam, The Netherlands, 2007, 341–347.

[Cra02] J.S. Cramer. *The Origins of Logistic Regression*. Tinbergen Institute Discussion Papers 02-119/4. Tinbergen Institute, 2002.

[CAB17] Antonia Creswell, Kai Arulkumaran und Anil A. Bharath. „On Denoising Autoencoders trained to minimise Binary Cross-entropy“. In: *CoRR* abs/1708.08487 (2017). arXiv: 1708.08487.

[Fri+12] V. Frinken, A. Fischer, R. Manmatha und H. Bunke. „A Novel Word Spotting Method Based on Recurrent Neural Networks“. In: *Transactions on Pattern Analysis and Machine Intelligence* 34.2 (2012), S. 211–224.

[GL15] Yaroslav Ganin und Victor Lempitsky. „Unsupervised Domain Adaptation by Backpropagation“. In: *International Conference on Machine Learning (ICML)*. Lille, France, 2015, 1180–1189.

[Gan+16] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario March und Victor Lempitsky. „Domain-Adversarial Training of Neural Networks“. In: *Journal of Machine Learning Research* 17.59 (2016), S. 1–35.

[Gio+17] Angelos P. Giotis, Giorgos Sfikas, Basilis Gatos und Christophoros Nikou. „A survey of document image word spotting techniques“. In: *Pattern Recognit.* 68 (2017), S. 310–332.

[GBB11] Xavier Glorot, Antoine Bordes und Yoshua Bengio. „Deep Sparse Rectifier Neural Networks“. In: Bd. 15. Machine Learning Research. Fort Lauderdale, FL, USA, 2011, S. 315–323.

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

[GD05] K. Grauman und T. Darrell. „The Pyramid Match Kernel: Discriminative Classification with Sets of Image Features". In: *International Conference on Computer Vision (ICCV)*. 2005.

[He+14] Kaiming He, Xiangyu Zhang, Shaoqing Ren und Jian Sun. „Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". In: *Lecture Notes in Computer Science* (2014), 346–361.

[He+16] Kaiming He, Xiangyu Zhang, Shaoqing Ren und Jian Sun. „Deep Residual Learning for Image Recognition". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

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

[Kan+20] Lei Kang, Marcal Rusinol, Alicia Fornes, Pau Riba und Mauricio Villegas. „Unsupervised Adaptation for Synthetic-to-Real Handwritten Word Recognition". In: *Winter Conference on Applications of Computer Vision (WACV)* (2020).

[KO11] Bekir Karlik und A Vehbi Olgac. „Performance analysis of various activation functions in generalized MLP architectures of neural networks". In: *International Journal of Artificial Intelligence and Expert Systems* 1.4 (2011), S. 111–122.

[KB17] Diederik P. Kingma und Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980.

[KJ16] Praveen Krishnan und C. V. Jawahar. *Matching Handwritten Document Images*. 2016. arXiv: 1605.05923.

[KJ19] Praveen Krishnan und C. V. Jawahar. „HWNet v2: An Efficient Word Image Representation for Handwritten Documents". In: *International Journal of Document Analysis and Recognition (IJDAR)* 22.4 (2019), S. 387–405.

[KSH12] Alex Krizhevsky, Ilya Sutskever und Geoffrey Hinton. „ImageNet Classification with Deep Convolutional Neural Networks". In: *Neural Information Processing Systems (NIPS)* 25 (2012).

[LT15] Aristomenis S Lampropoulos und George A Tsihrintzis. „Machine Learning Paradigms". In: *Applications İn Recommender Systems. Switzerland: Springer International Publishing* (2015).

[LRM04] V. Lavrenko, T. M. Rath und R. Manmatha. „Holistic word recognition for handwritten historical documents“. In: *First International Workshop on Document Image Analysis for Libraries*. 2004, S. 278–287.

[LSP06] S. Lazebnik, C. Schmid und J. Ponce. „Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories“. In: *Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. Bd. 2. 2006, S. 2169–2178.

[LeC+89] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard und L. D. Jackel. „Backpropagation Applied to Handwritten Zip Code Recognition“. In: 1.4 (1989), 541–551.

[LeC+98] Yann LeCun, Léon Bottou, Yoshua Bengio und Patrick Haffner. „Gradient-based Learning applied to Document Recognition“. In: *Proceedings of the IEEE* 86.11 (1998), S. 2278–2324.

[LKX] Fei-Fei Li, Ranjay Krishna und Danfei Xu. *Stanford Lecture Notes to CS231n: Convolutional Neural Networks for Visual Recognition*. Version: 2020. URL: http://cs231n.stanford.edu/.

[Low04] David Lowe. „Distinctive Image Features from Scale-Invariant Keypoints“. In: *International Journal of Computer Vision* 60 (2004), S. 91–.

[MHR96] R. Manmatha, Chengfeng Han und E.M. Riseman. „Word spotting: a new approach to indexing handwriting“. In: *Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. 1996.

[MB02] U.-V Marti und H Bunke. „The IAM-database: an English sentence database for offline handwriting recognition“. In: *Journal on Document Analysis and Recognition* 5 (2002), 39–46.

[MP43] Warren Mcculloch und Walter Pitts. „A Logical Calculus of Ideas Immanent in Nervous Activity“. In: *Bulletin of Mathematical Biophysics* 5 (1943), S. 127–147.

[Meh+19] Arpan R. Mehta, Puja R. Mehta, Stephen P. Anderson, Barbara L. H. MacKinnon und Alastair Compston. „Etymology and the neuron(e)“. In: *Brain* 143.1 (2019), S. 374–379.

[MP69] Marvin Minsky und Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press, 1969.

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

[RZL17] Prajit Ramachandran, Barret Zoph und Quoc V. Le. „Searching for Activation Functions“. In: *CoRR* abs/1710.05941 (2017).

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

[RS09] Jose Rodriguez-Serrano. „Handwritten word-spotting using hidden Markov models and universal vocabularies“. In: *Pattern Recognition* 42 (2009), S. 2106–2116.

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

[RHW86] David E. Rumelhart, Geoffrey E. Hinton und Ronald J. Williams. „Learning representations by back-propagating errors“. In: *Nature* 323.6088 (1986), S. 533–536.

[Rus+18a] Eugen Rusakov, Leonard Rothacker, Hyunho Mo und Gernot A. Fink. „A Probabilistic Retrieval Model for Word Spotting Based on Direct Attribute Prediction“. In: *International Conference on Frontiers in Handwriting Recognition (ICFHR)*. Niagara, USA, 2018, S. 38–43.

[Rus+18b] Eugen Rusakov, Sebastian Sudholt, Fabian Wolf und Gernot A. Fink. „Exploring Architectures for CNN-Based Word Spotting“. In: (2018). arXiv: 1806.10866.

[Rus+11] M. Rusinol, D. Aldavert, R. Toledo und J. Llados. „Browsing Heterogeneous Document Collections by a Segmentation-Free Word Spotting Method“. In: *International Conference on Document Analysis and Recognition*. 2011, S. 63–67.

[Rus+15] Marçal Rusiñol, David Aldavert, Ricardo Toledo und Josep Lladós. „Efficient segmentation-free keyword spotting in historical document collections“. In: *Pattern Recognition* 48 (2015).

[RN09] Stuart Russell und Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. USA, 2009.

[Sha17] Sagar Sharma. „Activation functions in neural networks“. In: *Towards Data Science* 6 (2017).

[SZ14] Karen Simonyan und Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. arXiv: 1409.1556.

[SF16] Sebastian Sudholt und Gernot A. Fink. „PHOCNet: A Deep Convolutional Neural Network for Word Spotting in Handwritten Documents“. In: *International Conference on Frontiers in Handwriting Recognition (ICFHR)* (2016).

[SF17] Sebastian Sudholt und Gernot A. Fink. „Evaluating Word String Embeddings and Loss Functions for CNN-Based Word Spotting“. In: *International Conference on Document Analysis and Recognition (ICDAR)*. Bd. 01. 2017, S. 493–498.

[SF18] Sebastian Sudholt und Gernot A. Fink. „Attribute CNNs for Word Spotting in Handwritten Documents“. In: *International Journal on Document Analysis and Recognition (IJDAR)* 21.3 (2018), S. 159–160.

[SGF17] Sebastian Sudholt, Neha Gurjar und Gernot A. Fink. „Learning Deep Representations for Word Spotting Under Weak Supervision“. In: (2017). arXiv: 1712.00250.

[WKW16] Karl R. Weiss, T. Khoshgoftaar und Dingding Wang. „A survey of transfer learning“. In: *Journal of Big Data* 3 (2016), S. 1–40.

[WBF20] Fabian Wolf, Kai Brandenbusch und Gernot A. Fink. „Improving Handwritten Word Synthesis for Annotation-free Word Spotting“. In: *International Conference on Frontiers in Handwriting Recognition (ICFHR)*. To appear. 2020.

[WF20] Fabian Wolf und Gernot A. Fink. „Annotation-free Learning of Deep Representations for Word Spotting using Synthetic Data and Self Labeling“. In: *International Workshop on Document Analysis Systems*. Wuhan, China, 2020. arXiv: 2003.01989.

[Yeg04] B. Yegnanarayana. *Artificial Neural Networks*. Prentice-Hall of India Pvt.Ltd, 2004.

[ZC88] Zhou und Chellappa. „Computation of optical flow using a neural network“. In: *International Conference on Neural Networks*. Bd. 2. 1988, S. 71–78.

[ZG09] Xiaojin Zhu und Andrew B. Goldberg. *Introduction to Semi-Supervised Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. 2009.