TECHNISCHE UNIVERSITÄT DRESDEN

ZENTRUM FÜR INFORMATIONSDIENSTE
UND HOCHLEISTUNGSRECHNEN
PROF. DR. WOLFGANG E. NAGEL

Master-Arbeit

zur Erlangung des akademischen Grades
Master of Science

Anwendung neoronaler Netze zur inhaltsbasierten Bildsuche bei historischen Bilddarstellungen

Jan Philipp Simon Langen
(Geboren am 26. Dezember 1994 in Münsterlingen)

Hochschullehrer: Prof. Dr. Wolfgang E. Nagel
Betreuer: Dr. Christoph Lehmann & Dr. Taras Lazariv

Dresden, 3. August 2020

Aufgabenstellung der Masterarbeit

Name des Studenten: Langen, Jan Philipp Simon

Matrikelnummer: 4081562

Studiengang: Master Informatik TU Dresden

Thema: **Anwendung neuronaler Netze zur inhaltsbasierten Bildersuche bei historischen Bilddarstellungen auf Basis des DELF-Ansatzes (DEep Local Feature)**

Zielstellung: Aufgrund oftmals unvollständiger Metadaten und der großen Heterogenität historischer Bilddarstellungen ist die inhaltsbasierte Bildsuche bei der Nutzung großer Bilddatenbanken ein wichtiges Werkzeug. Gegenüber herkömmlichen Methoden (z.B. RANSAC, SIFT) liefern Ansätze basierend auf neuronalen Netzen vielversprechende Ergebnisse. Dabei ist DELF eine der aktuellen Entwicklungen. In der Masterarbeit werden folgende Punkte behandelt:

1. Anwendung und Dokumentation der verfügbaren Implementierung von Delf speziell zur inhaltsbasierten Suche in historischen Bildern
2. Realisation der Implementierung auf PowerPC auf Taurus
3. Abgrenzung und Vergleich von DELF mit mindestens einem weiteren Ansatz
4. Experimente zur Parametergestaltung von DELF für historische Bilder

Betreuender Hochschullehrer: Prof. Dr. Wolfgang E. Nagel

Betreuer: Dr. Christoph Lehmann, Dr. Taras Lazariv

Beginn: 22.06.2020

Abgabe: 23.11.2020

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tag dem Prüfungsausschuss der Fakultät Informatik eingereichte Master-Arbeit zum Thema:

Anwendung neoronaler Netze zur inhaltsbasierten Bildsuche bei historischen Bilddarstellungen

vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Dresden, den 3. August 2020

Jan Philipp Simon Langen

Kurzfassung

Abstract

Inhaltsverzeichnis

1 Motivation **2**

2 Verwandte Arbeiten **4**

3 DELF **8**

3.1 ResNet 9
3.2 Trainingsdaten 11
3.3 Fine-Tuning 11
3.4 Attention Training 12
3.5 Extraktion und Verarbeitung 14
3.5.1 Multi-Skalen-Extraktion 14
3.5.2 Deskriptorlokalisierung 14
3.5.3 Deskriptorselektion 16
3.5.4 Datentransformation und Dimensionsreduktion 17
3.6 Matching 18
3.7 Verfahrensunterschiede im Delf Artikel 20

4 Parameter Analyse **22**

4.1 Hyperparameteroptimierung der Trainingsphasen 22

Literaturverzeichnis **30**

1 Motivation

Präzise und effiziente Suchwerkzeuge sind essenziell um große Datenmengen für einen Nutzer sinnvoll verwertbar zu machen. Dies gilt insbesondere auch im Bereich der Bildersuche. Die klassische Bildersuche basiert auf vom Nutzer formulierten Anfragen, mit deren Hilfe das Suchsystem eine Liste an passenden Bildkandidaten zusammenstellt und zurückgibt. Hierbei nutzt das System eine Reihe von Zusatzinformationen, sogenannten Metadaten, wie Tags, Titel, Aufnahmeort oder Datum. Eine alternativer Ansatz der Suche, der in dieser Arbeit behandelt wird, ist die inhaltsbasierte Bildersuche (engl. Content-Based Image Retrieval, kurz CBIR). Hierbei werden vom Nutzer keine Anfragen formuliert. Stattdessen dient ein Bild als Suchanfrage. Ziel ist es, Bilder mit gleichem oder ähnlichem Bildinhalt als Ergebnis zurückzugeben. Ein Vorteil dieser Herangehensweise ist, dass der Nutzer keine Informationen über den Inhalt des Suchbildes benötigt. Das System arbeitet ausschließlich mit den Pixelinformationen der Bilder. Ein weiterer Vorteil ist daher, dass weder im Suchbild noch in der Suchdatenbank Metadaten zu den Bildern vorhanden sein müssen, was die Einsatzmöglichkeiten von inhaltsbasierter Suche sehr flexibel gestaltet. Im Folgenden wird die inhaltsbasierte Suche auch als Image Retrieval bezeichnet.

Das Anwendungsgebiet dieser Arbeit ist die Suche auf historischen Bildern. Diese weit gefasste Domäne ist besonders herausfordernd, da sie sehr heterogene Daten enthält. Dabei gibt es nicht nur große Unterschiede in den abgebildeten Bildinhalten, wie Gebäude, Naturaufnahmen oder Portraits, sondern auch in den verwendeten Aufnahmeverfahren. Durch die Fortschritte der Aufnahmetechnik können historische Bilder sowohl in Form von Zeichnungen oder Malerei, aber auch als Druck oder in anfänglichen Formen der Photographie vorliegen. Da Metadaten zu historischen Bildern erst bei der Digitalisierung hinzugefügt werden können, sind diese oft gar nicht oder nur lückenhaft vorhanden. Dies macht die inhaltsbasierte Suche für diese Domäne im Vergleich zur klassischen Bildersuche zu einer besonders geeigneten Methode. Mit der Umsetzung einer unterstützenden Suche für das UrbanHistory4D Projekt [1] ergibt sich ein konkreter Anwendungsfall für diese Arbeit. Das UrbanHistory4D Projekt befasst sich mit der Erstellung interaktiver Stadtkarten. Wo vorhanden kann sich der Nutzer historische Aufnahmen anzeigen lassen, die den Ort zeigen, an dem er sich innerhalb der Karte befindet. Das Akkumulieren und Zuordnen von historischen Bildern zu diesen Plätzen ist ein wesentlicher Arbeitsanteil bei der Erstellung der Karten. Image Retrieval Systeme können helfen den Suchaufwand für die Ersteller der Karten signifikant zu reduzieren. Dabei handelt es sich um einen aktiven Forschungsbereich, in dem momentan unterschiedliche Suchsysteme analysiert werden.

Das Image Retrieval Verfahren DELF (attentive DEep Local Features) [2] entwickelt von Noh, Araujo et al., welches in dieser Arbeit untersucht wird ist ein Deep Learning Ansatz. Durch den raschen Fortschritt im Bereich tiefer neuronaler Netzwerkarchitekturen der letzten Jahre erfreuen sich gelernte Ansätze immer größerer Beliebtheit. DELF erzielt auf bekannten Benchmarkdatensätzen wie Oxford5k [3] und Paris6k [4] sehr gute Ergebnisse. Besonders gut schneidet DELF im Vergleich auf dem eigens erstellten

Google Landmarks Datensatz [5] ab. Dieser enthält mit über 1 Mio. Bilder und 13k unterschiedlichen Motiven eine deutlich heterogenere Mischung an Objekten als andere Benchmarks. Die gute Performanz auf diesem Datensatz lässt also hoffen, dass sich das DELF-Verfahren auch für die historische Domäne eignet.

2 Verwandte Arbeiten

Bei Information Retrieval handelt es sich um ein Problem aus dem Bereich der Computer Vision, welches bereits seit langem intensiv erforscht wird. In frühen Ansätzen versuchte man vor allem globale Beschreibungen von Bildern zu erstellen, um diese untereinander vergleichen zu können. Diese basierten zum Beispiel auf Farbhistogrammen oder Texturbeschreibungen [6]. Allerdings waren diese Ansätze oft sehr anfällig für Unterschiede in Beleuchtung, Skalierung und anderen Transformationen, wie sie bei unterschiedlichen Aufnahmen des selben Motivs auftreten können.
Ein wesentlicher Durchbruch gelang David G. Lowe 2004 mit der Entwicklung des SIFT-Verfahrens (Scale Invariant Feature Transform) [7]. Hierbei werden mehrere Konzepte vereint um Bildbeschreibungen zu erzeugen, die robuster gegenüber unterschiedlichen Transformationen sind. So arbeitet der SIFT Algorithmus beispielsweise nicht direkt auf den Bildern, sondern im sogenannten Scale Space. Dieser besteht auf unterschiedlich skalierten Versionen des Ursprungsbildes, auf welche wiederum unterschiedlich starke Gauß-Filter angewendete werden. Betrachtet werden schließlich Differenzbilder zwischen benachbarten Stärken der Gauß-Fitler Ergebnisse. Die Verwendung von unterschiedlich skalierten Bildversionen macht die berechneten SIFT-Merkmale deutlich robuster gegen Skalierungsunterschiede. Das SIFT-Verfahren besteht aus zwei Phasen. In der ersten Phase werden über die Suche nach lokalen Extrema bedeutsame Bildpunkte ausgewählt. Für diese werden in der zweiten Phase einzelne Deskriptoren berechnet. Das Bild wird also nicht global beschrieben, sondern über viele lokale Deskriptoren dargestellt. Die lokalen Deskriptoren ergeben sich aus Histogrammen der Gradientrichtungen umliegender Bildpunkte. Diese werden relativ zu der dominanten Gradientrichtung in der Umgebung berechnet, was die Deskriptoren invariant gegenüber Rotationen macht. Lowes Entwicklung bildet den Ursprung für viele abgeleitete Verfahren wie SURF[8], PCA-SIFT[9] und RIFT[10]. Auch in aktueller Forschung werden Image Retrieval Verfahren untersucht, die mit SIFT-Merkmalen arbeiten [11].
Der Trend bei der Entwicklung neuer Image Retrieval Systeme geht aktuell jedoch hauptsächlich in Richtung von gelernten Verfahren. Die Basis dieser Verfahren bilden tiefe CNN-Architekturen (Convolutional Neural Networks). Ein neuronales Netzwerk lässt sich als eine schichtweise Aneinanderreihung nichtlinearer Funktionen auffassen. Convolutional Neural Networks sind ein Sonderfall neuronaler Netze, welche sogenannte Convolutional Layer, zu deutsch faltende Schichten, enthalten. In diesen Schichten werden Faltungs/ bzw. Filteroperationen auf die Eingabedaten angewendet, um für das Netzwerk hilfreiche Merkmale in den Daten hervorzuheben. Dies ist durchaus vergleichbar mit den Filteroperationen, die im SIFT-Verfahren verwendet werden. Der Unterschied besteht jedoch darin, dass die Parameter der verwendeten Filtermasken sowie aller anderen Netzparameter nicht per Hand gewählt, sondern in einem Trainingsverfahren für den aktuellen Anwendungsfall optimiert werden. Der Entwickler bestimmt lediglich die grobe Architektur des Netzwerks, also die Anzahl, Größe und Reihenfolge der verwendeten Schichten sowie die Art der Operationen, die in ihnen durchgeführt werden. In Image Retrieval Systemen werden CNNs eingesetzt, um Bilddeskriptoren zu erstellen. Hierfür werden Zwischenergebnisse des Netzwerks, also die Ausgaben einer bestimmten Schicht genutzt. An welcher Stelle im Netzwerk die

Deskriptoren entnommen werden ist dabei von entscheidender Bedeutung. Zeiler und Fergus haben in ihrer Studie zur Visualisierung von CNNs gezeigt [12], dass die früheren Schichten von CNNs typischerweise einfache Konzepte wie Kanten oder Ecken hervorheben. Mit wachsender Tiefe der betrachteten Netzwerkschicht steigt auch die Komplexität der Konzepte, die von den Ausgaben der Schicht beschrieben werden können.

In dem in [13] beschriebenen Image Retrieval System von Babenko, Slesarev et al. wird als Modell ein CNN bestehend aus fünf faltenden gefolgt von drei voll verbundenen Schichten (im Englischen fully-connected layer) genutzt. Als Deskriptoren werden die Ausgaben der ersten bzw. zweiten voll verbundenen Schicht verwendet. In einer voll verbundenen Schicht hat jeder Wert der Eingabe Einfluss auf jeden Wert in der Ausgabe. Die Ausgaben solcher Schichten werden also von der gesamten Bildeingabe beeinflusst und können daher als globale Deskriptoren verstanden werden. Diese intuitive Herangehensweise erzielt leichte Verbesserung gegenüber den zur Zeit der Veröffentlichung gängigen algorithmischen Verfahren.
Razavian, Sullivan et al. stellen in [14] ein System auf Basis der in [15] beschriebenen Netzwerkarchitektur vor. Das Modell besteht ebenfalls aus fünf faltenden und drei voll verbundenen Schichten. Die Deskriptoren stammen aus den Ausgaben der letzten faltenden Schicht. Anders als bei Babenko, Slesarev et al. werden in diesem System mehrere Deskriptoren pro Bild erstellt. Hierfür werden systematisch Teilbilder aus Bildbereichen unterschiedlicher Größe generiert. Anschließend werden die Teilbilder auf eine feste Größe skaliert und als Eingabe in das Netzwerk gegeben. So wird für jeden betrachteten Bildbereich ein eigener lokaler Deskriptor erstellt. In ihren Experimenten stellen die Autoren fest, dass die Verwendung von lokalen Deskriptoren gegenüber einer globalen Betrachtung zu einer signifikanten Verbesserung der Retrievalperformanz führt. Der überwiegende Teil aktueller Retrieval Systeme setzt auf die Erstellung von lokalen Deskriptoren.
Eine interessante Frage bei der Konzeption von Image Retrieval Systemen, die mit lokalen Deskriptoren arbeiten ist, wie man entscheidet, welche Bildregionen am sinnvollsten zu betrachten sind. Das ONE-Verfahren [16] von Xie, Hong et al. nutzt ein VGG-19 [17] Modell und extrahiert Deskriptoren aus der vorletzten voll verbundenen Schicht. Als Eingaben in das Netzwerk dienen sogenannte Object Proposals. Dabei handelt es sich um Bildausschnitte, welche Regionen umschließen, in denen Objekte vermutet werden. Die Autoren testen sowohl manuell annotierte sowie automatisch extrahierte Object Proposals und erzielen mit beiden Ansätzen ähnlich gute Ergebnisse. Für die automatische Bestimmung von Object Proposals nutzen sie das Selective Search Verfahren [18].
Das Delf-Verfahren [2], welches in dieser Arbeit untersucht wird, basiert ebenfalls auf lokalen Deskriptoren. Die Deskriptoren werden aus einer faltenden Schicht aus dem hinteren Teil eines ResNet-50 [19] Modells extrahiert. Als Eingabe in das Netzwerk werden Bilder in ihrer Gesamtheit betrachtet. Da bis zur Extraktionsschicht keine voll verbundenen Schichten genutzt werden, kann für jeden extrahierten Wert zurückgerechnet werden, von welchen Bereichen des Ursprungsbildes er beeinflusst wurde. Dies erlaubt es die Ausgaben der Extraktionsschicht in einzelne lokale Deskriptoren zu unterteilen. Um auszuwählen welche der lokalen Deskriptoren zur Darstellung eines Bildes genutzt werden sollen, werden die lokalen Deskriptoren in ein weiteres neuronales Netz gegeben. Dieses Netz hat die Aufgabe zu bewerten, wie geeignet die einzelnen Deskriptoren zur Beschreibung des Gesamtbildes sind. Auf Grund dieser Bewertung werden die wichtigsten Deskriptoren zu jedem Bild ausgewählt, wogegen schlecht bewertete Deskripto-

ren verworfen werden. Der konzeptionelle Unterschied bei der Auswahl der Deskriptoren im Vergleich zum ONE-Verfahren ist, dass die Auswahl auf Grund der bereits berechneten Deskriptoren geschieht anstatt auf Grund des Ursprungsbildes. Die Funktionsweise des Delf-Verfahrens wird in Kapitel 3 ab Seite 8 im Detail erklärt.
Bevor Neuronale Netze für die Erstellung von Deskriptoren genutzt werden können, müssen ihre Parameter in einem Trainingsverfahren optimiert werden. Während dem Training muss das Netzwerk eine Aufgabe lösen. Wie erfolgreich das Netzwerk dabei ist, wird mit Hilfe einer Fehlerfunktion dargestellt. Das Netz versucht seine Parameter so anzupassen, dass die Fehlerfunktion minimiert wird. Im Fall der bereits vorgestellten Verfahren wird dabei eine Dummy-Aufgabe, typischerweise die Klassifikation von Bildern, gelöst. In der letzten Zeit wurden jedoch einige Ansätze veröffentlicht, die versuchen neuronale Netze direkt an Image Retrieval Aufgaben zu trainieren. Radenović, Tolias und Chum stellen in [20] eine solchen Ansatz vor. Während dem Training betrachten sie dabei Bildpaare. Bei diesen Paaren handelt es sich entweder um korrekte Matches, falls die Bilder ähnliche Bildinhalte darstellen, oder um inkorrekte Matches, falls dies nicht der Fall ist. Die Bilder eines Paares durchlaufen identische Netze und erzeugen dabei jeweils eine Ausgabe. Für die Optimierung wird eine spezielle Fehlerfunktion definiert, welche bewertet wie stark sich die Netzwerkausgaben der Paare voneinander unterscheiden. Für korrekte Matches sollten sich die Netzwerkausgaben möglichst ähneln. Wird jedoch ein inkorrektes Match betrachtet, sollten auch die Ausgaben eine Mindestdifferenz zueinander aufweisen. Die Autoren testen ihr Verfahren auf unterschiedlichen CNN Architekturen wie VGG [17] und AlexNet [21] und erzielen damit sehr gute Ergebnisse auf gängigen Retrievalbenchmarks. Das direkte Training auf Retrievalaufgaben ist eine vielversprechende neue Forschungsrichtung im Retrievalbereich, an der momentan intensiv gearbeitet wird.

Da Image Retrieval Systeme meist auf großen Bilddatenbanken eingesetzt werden und somit für eine Suchanfrage viele Bilder miteinander verglichen werden müssen, ist es sinnvoll Bildrepräsentationen so kompakt wie möglich zu gestalten, um die Laufzeit der Suche zu verbessern. Insbesondere bei Verfahren, die lokale Deskriptoren erstellen und häufig hunderte oder tausende Merkmale pro Bild extrahieren, kann mit einer guten Kodierung viel Rechenzeit gespart werden. Ein beliebter Ansatz zur Erstellung kompakter Darstellungen aus lokalen Deskriptoren ist das BOVW-Modell (Bag-of-Visual-Words) [22], erstmals vorgestellt im Kontext von Textklassifikation von McCallum und Nigam. Hierbei werden zunächst alle aus einem Datensatz extrahierte Deskriptoren mittels Clusteranalyse (bspw. K-Means-Clustering [23]) in Gruppen eingeteilt. Deskriptoren, die dem gleichen Cluster zugeordnet werden, werden dabei auf das selbe "visuelle Wort" abgebildet. Als Beschreibung des Gesamtbilds dient ein Histogramm über die im Bild enthaltenen visuellen Wörter. Bei diesem Verfahren geht durch Quantisierung ein Teil der Information verloren. Das ebenfalls auf Clustering basierte VLAD-Verfahren [24] von Jégou, Douze et al. versucht diese Information nutzbar zu machen, indem es statt der Vorkommen die Quantisierungsfehler akkumuliert, die beim abbilden auf die nächsten visuellen Worte entstehen.
Um eine Suchanfrage mit einer Rangliste der ähnlichsten Bilder zum Suchbild beantworten zu können, werden die Deskriptoren der Bilder in der Datenbank mit denen des Suchbildes verglichen. Als Metrik dient hierbei meist die euklidische Distanz. Auf kleinen Datensätzen ist es laufzeittechnisch sinnvoll alle Kombinationen von Such- und Datenbankbildern zu vergleichen. Häufig werden bei größeren Datensätzen jedoch Methoden der approximierten nächsten Nachbarsuche (ANN) verwendet. Diese garantieren

zwar kein optimales Ergebnis, erlauben jedoch deutlich schnellere Verarbeitung von Suchanfragen. So gibt es zum Beispiel Ansätze Deskriptoren mit Hilfe spezieller Hashfunktionen zu vergleichen. Diese werden so konstruiert, dass ähnliche Deskriptoren auf die gleichen bzw. möglichst ähnliche Hashcodes abgebildet werden, während gleichzeitig die Kollisionswahrscheinlichkeit für sehr unterschiedliche Deskriptoren minimal gehalten wird. Wang et al. beschreiben in ihrer Studie [25] unterschiedliche Konzepte für die Erstellung solcher Hashfunktionen.

3 DELF

Das Delf-Verfahren [2] von Noh, Araujo et al. bildet die Basis für die Experimente, die in dieser Arbeit durchgeführt werden. In dem zu Delf veröffentlichten Artikel werden einige Verfahrensschritte nicht, oder nur oberflächlich beschrieben, was als Grundlage für eine detailgetreue Neuimplementierung nicht ausreicht. Die Autoren stellen ebenfalls den Quellcode einer Implementierung des Delf-Verfahrens zur Verfügung[1]. Dieser weist jedoch einige konzeptionelle Unterschiede zu dem im Artikel beschriebenen Vorgehen auf. Für den Experimentalteil dieser Arbeit wurde Delf neu implementiert[2]. Diese Implementierung orientiert sich an dem offiziell veröffentlichten Quellcode und wird im folgenden Kapitel schrittweise im Detail erklärt. Die Unterschiede zu den Beschreibungen des veröffentlichten Artikels werden in den letzten Abschnitten diese Kapitels ab Seite 20 erörtert.

Das Delf-Verfahren lässt sich in vier Phasen einteilen (siehe Abb. 3.1). Zu Beginn steht das sogenannte Fine-Tuning. Hierbei wird ein vortrainiertes Modell, in unserem Fall ein ResNet-50 Netzwerk, auf einem neuen Datensatz weiter trainiert. Die Domäne der Bilder dieses Datensatzes sollte dabei möglichst nahe der späteren Retrievalaufgabe sein, damit das Modell lernen kann aussagekräftige Deskriptoren für diese Art von Bilden zu berechnen. In der nächsten Phase wird auf dem Modell aufbauend ein Attention-Netzwerk trainiert welches die Güte berechneter Deskriptoren bewertet. In der dritten Phase werden für die Bilder der Datenbank, in der gesucht werden soll Deskriptoren extrahiert. Mit Hilfe des Attention-Netzwerks wird eine Vorauswahl besonders geeigneter Deskriptoren getroffen. In anschließenden Vorverarbeitungsschritte werden die Deskriptoren in den Bildern lokalisiert und in eine kompaktere Form überführt. In der finalen Phase kann Delf aktiv genutzt werden. Es können nun Bilder als Suchanfragen gestellt werden. Delf vergleicht eine Anfrage mit allen Bildern des Datensatzes anhand der vorverarbeiteten Deskriptoren. Potentielle Matches zwischen Deskriptoren werden in einem letzten Schritt geometrisch verifiziert. Das Ergebnis einer Anfrage ist eine Rangliste der ähnlichsten Bilder, sortierte nach der Anzahl verifizierte Deskriptoren-Matches mit dem Anfragebild.

[1] `https://github.com/tensorflow/models/tree/master/research/delf`, zuletzt besucht 16.06.20
[2] `https://gitlab.hrz.tu-chemnitz.de/s5407552--tu-dresden.de/hist4delf`

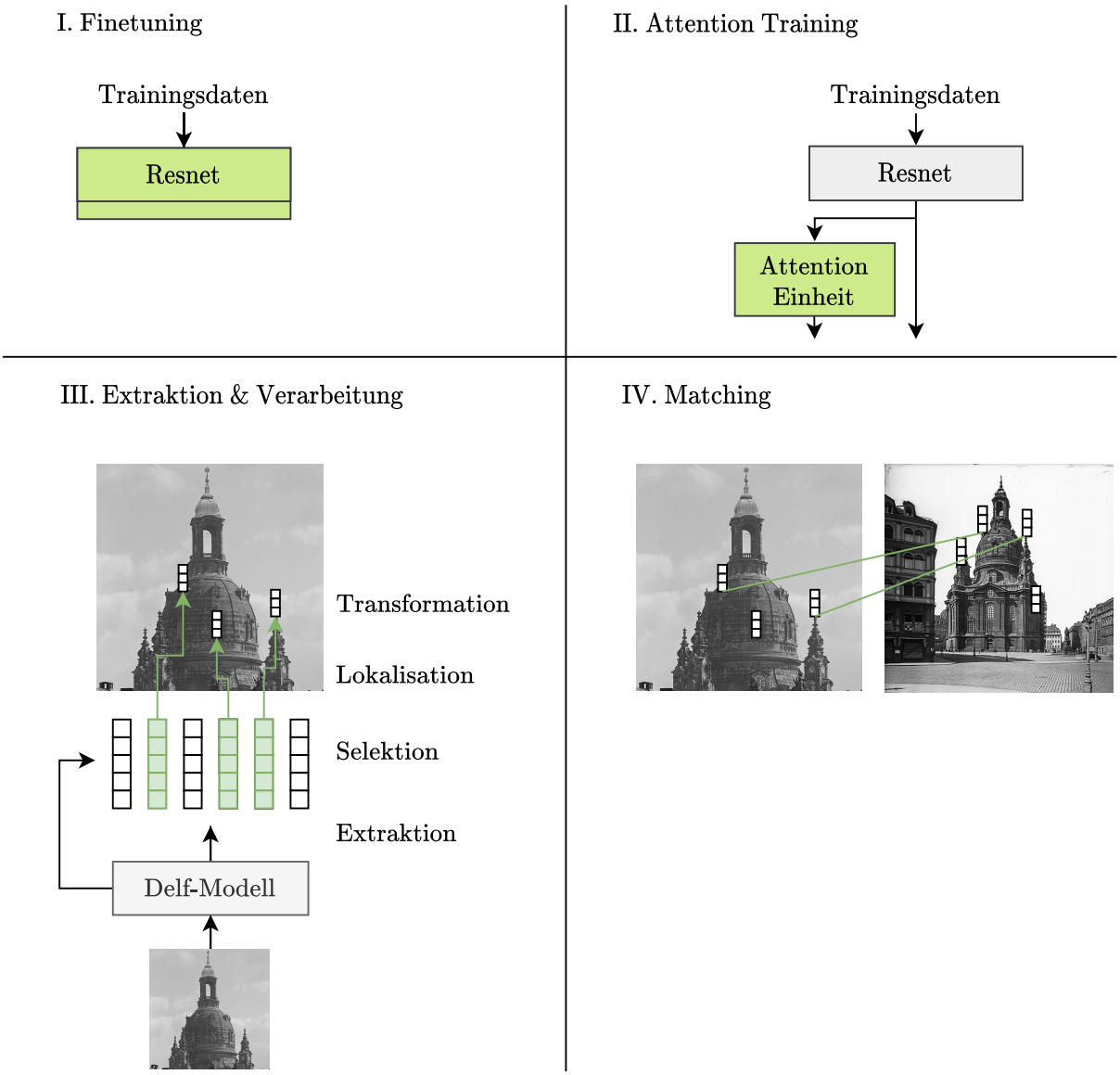


Abbildung 3.1: Die Phasen des Delf-Verfahrens

3.1 ResNet

Das Delf-Verfahren nutzt zur Erstellung von Deskriptoren ein Residuales Netzwerk (kurz ResNet). Bei der im Jahre 2015 vorgestellten ResNet Architektur [19] von He, Zhang et al. handelt es sich um eine der meist genutzten tiefen CNN-Architekturen der aktuellen Forschung. ResNets finden Anwendung in unterschiedlichen Machine Learning Aufgaben, wie Klassifikation, Objektdetektion oder Image Retrieval. Zeiler und Fergus haben gezeigt [12], dass CNNs mit wachsender Netzwerktiefe in der Lage sind komplexere Merkmale zu detektieren. Es scheint daher intuitiv zur Lösung immer komplexerer Aufgaben zunehmend tiefere Netzwerke zu konstruieren. Allerdings stellt sich heraus, dass ab einem gewissen Punkt keine Verbesserungen mehr mit dem bloßen aneinanderreihen von immer mehr Schichten erzielt werden können. Werden zu viele Schichten hinzugefügt kann es sogar passieren, dass die Qualität der Netz-

werkvorhersagen abnimmt. Mit dem rasanten Anstieg der Anzahl an Netzwerkparameter wird es immer schwieriger das Netzwerk zu optimieren. Parameter konvergieren deutlich langsamer zu einem Optimum und es gibt mehr lokale Minima in denen ein Netzwerk im Optimierungsprozess stecken bleiben kann. ResNets wirken diesem Problem mit der Einführung sogenannter Skip-Verbindungen entgegen. Hierbei werden zusätzliche Direktverbindungen im Netzwerk geschaffen, bei denen einige Schichten übersprungen werden. Fließt eine Eingabe an den Beginn einer Skip-Verbindung, so wird auf dieser die Identität der Eingabe mitgeführt. Parallel durchläuft die Eingabe die übersprungenen Schichten. Am Ausgangspunkt der Verbindung wird schließlich die Ausgabe der übersprungenen Schichten mit der Identität summiert (siehe Abb. 3.2a). Durch die Bereitstellung der Identität hat das Netzwerk eine bessere Grundlage zur Optimierung und einzelne schlecht optimierte Schichten weniger negative Auswirkung auf die Netzwerkausgabe. Die Autoren stellen fest, dass CNNs bei Verwendung von Skip-Verbindung schneller zu einem Optimum konvergieren und dabei bessere Minima gefunden werden.

ResNets können in unterschiedlichen Konfigurationen erstellt werden. Das für Delf verwendete ResNet-50

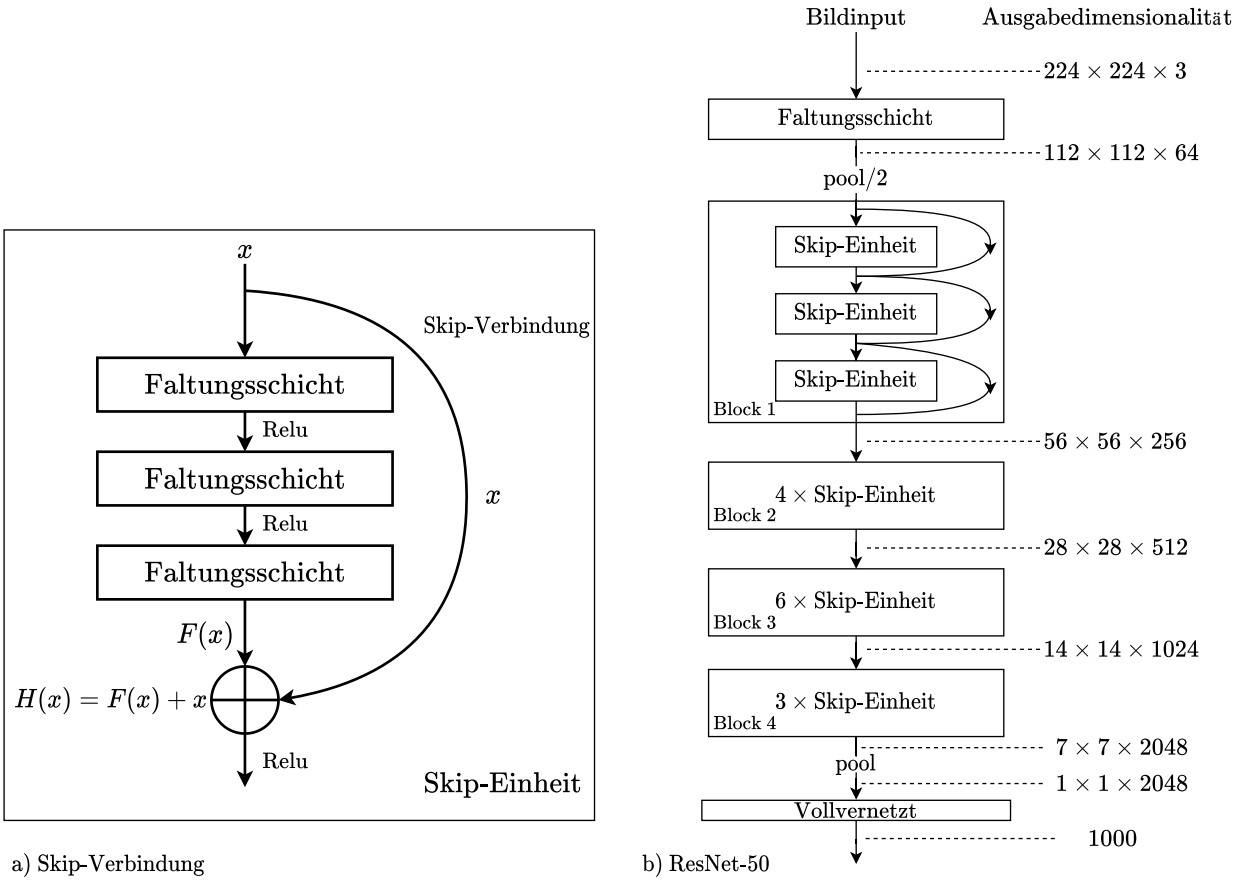


Abbildung 3.2: Aufbau der ResNet-Architektur (vgl. Fig.2, Fig.3 aus [19])

besteht aus 49 faltenden gefolgt von einer vollvernetzen Schicht. Skip-Verbindungen überspringen jeweils drei Schichten. Das Netzwerk kann in vier Blöcke unterteilt werden. Die Größe der einzelnen Featuremaps, der Ausgabe verringert sich nach jedem Block um den Faktor vier, wohingegen die Merkmalstiefe bzw. Anzahl der Featuremaps in der Ausgabe steigt (Siehe Abb. 3.2b). In der Implementierung

dieser Arbeit wird die von Torchvision zur Verfügung gestellte ResNet-50 Architektur genutzt[3].

3.2 Trainingsdaten

Um die Modelle für das Delf-Verfahren zu trainieren wird ein gelabelter Datensatz benötigt. Zum jetzigen Zeitpunkt steht kein solcher Datensatz von historischen Stadtaufnahmen mit ausreichender Größe zur Verfügung. Für diese Arbeit wird daher alternativ auf die Bilder der Google Landmark Challenge V2 [26] zurückgegriffen. Die Bilder entstammen einer Websuche auf der Wikimedia Datenbank[4] und zeigen Sehenswürdigkeiten aus der ganzen Welt. Der überwiegende Teil (72%) zeigt dabei menschengemachte Sehenswürdigkeiten, wie Kirchen, Museen oder Häuser. Auch wenn historische Aufnahmen keinen wesentlichen Teil der Bilder ausmachen enthält der Datensatz viele ähnliche Inhalte zu den historischen Datensätzen, die für das Retrieval genutzt werden. Der Trainingssatz der Landmark Challenge ist mit über 4 Millionen Bildern aus über 200k unterschiedlichen Kategorien sehr groß und heterogen. Da bei der Zusammenstellung keine Verifizierung der Bildinhalts durchgeführt wird kommt es häufig vor, dass Bilder in der falschen Kategorie einsortiert sind. Für das Delf-Training wird daher ein bereinigtes Subset des Datensatzes verwendet, welches von Yokoo, Ozaki et al. in Rahmen ihrer Arbeit [27] erstellt wurde. Aus dem bereinigten Datensatz werden die 40 häufigsten Kategorien gewählt und so ein Trainingsdatensatz über 39 790 gelabelten Bildern erstellt.

3.3 Fine-Tuning

Das Ziel der ersten Trainingsphase ist es das ResNet Modell so zu optimieren, dass das Modell bei der Verarbeitung eines Bildes Zwischenergebnisse erzeugt, die den Bildinhalt aussagekräftig beschreiben. Dies ist die Voraussetzung, um später leistungsstarke Deskriptoren erstellen zu können. Während der Optimierung versucht das Netzwerk die Klassifikationsaufgabe des Trainingsdatensatzes zu lösen. Zu Beginn werden die Netzwerkparameter dabei nicht zufällig initialisiert. Stattdessen wird ein Vortrainiertes Modell als Ausgangspunkt genutzt. Dieser Ansatz des Transferlernens wird als Fine-Tuning bezeichnet und ist eine gängige Methode, um den Trainingsprozess zu erleichtern. Auch in anderen Image Retrival Systemen [14] [20] wird diese Methode für die Netzwerkoptimierung genutzt. Zeiler und Fergus zeigen in ihren Experimenten (vgl. [12] Kapitel 5.2), dass Netzwerke beim Training lernen allgemein nützliche Merkmale zu extrahieren, die sich auf unterschiedliche Datensätze anwenden lassen. Um ein vortrainiertes Netzwerk auf einen neuen Datensatz anzupassen sind daher nur kleine Veränderungen der Netzwerkparameter notwendig.

Als Ausgangspunkt für das Delf-Training wird ein auf ImageNet trainiertes ResNet-50 genutzt. Bei ImageNet handelt es sich um einen sehr große Klassifikationsdatensatz mit 1.4M Bildern aus 1000 sehr unterschiedlichen Kategorien. Durch die Vielfalt an Kategorien eigenen sich auf ImageNet trainierte Netzwerke als Ausgangspunkt für viele Klassifikationsaufgaben. Daher stellen die meisten Machine Learning Frameworks auf ImageNet trainierte Netzwerke zur Verfügung[5].

Während dem Training erwartet das Netzwerk quadratische Bilder mit einer Seitenlänge von 224 Pixeln

[3] `https://github.com/pytorch/vision/blob/c2e8a00885e68ae1200eb6440f540e181d9125de/torchvision/models/resnet.py`, zuletzt besucht 16.06.20

[4] `https://commons.wikimedia.org`, zuletzt besucht 18.06.20

[5] `https://pytorch.org/docs/stable/torchvision/models.html`, zuletzt besucht 23.06.20

und 3 Farbkanälen als Eingabe. Hierfür werden die Trainingsdaten zunächst quadratisch zugeschnitten und auf 250×250 Pixel skaliert. Anschließend wird ein zufälliger 224×244 Pixelbereich ausgewählt. Um das vortrainierte Netzwerk auf dem Trainingssatz weiter zu trainieren muss das Netzwerk so angepasst werden, dass die Ausgaben der letzten Schicht die korrekte Form für die zur Optimierung verwendete Fehlerfunktion hat. Als Fehlerfunktion wird hier die Kreuzentropie berechnet:

$$\text{Kreuzentropie}(Y, \hat{Y}) = -\sum_{c \in C} Y(c) * \log \hat{Y}(c) \tag{3.1}$$

$\hat{Y}$ ist hierbei die Verteilung der Klassenwahrscheinlichkeiten, die das Modell für eine Eingabe vorhergesagt hat. $\hat{Y}(c)$ ist die vom Netzwerk bestimmte Wahrscheinlichkeit, mit der die Eingabe der Klasse c zuzuordnen ist. Y beschreibt die tatsächliche Kategorie der Eingabe. Y ist also ein Verteilung, bei der die Wahrscheinlichkeit für jede, bis auf die korrekte Kategorie 0 und für die tatsächliche Klasse 1 ist. Als Ausgabe des Netzwerks wird also ein Vektor der Wahrscheinlichkeitsverteilung erwartet, dessen Dimensionalität der Anzahl der unterschiedlichen Klassen $|C|$ im Datensatz entspricht und dessen Einträge sich auf 1 summieren.
Damit die Ausgaben des Netzwerks die richtige Dimensionalität haben wird zunächst die letzte voll verbundene Schicht entfernt. An ihre Stelle tritt eine faltenden Schicht mit 1×1 Filtermasken, die eine Merkmalstiefe von 2048 erwarten, was der Dimensionalität vorangehenden Schicht entspricht (vgl. Abb. 3.2b). In der faltenden Schicht werden $|C|$ dieser Filtermasken auf die Eingabe angewendet, wodurch die Ausgabe die gewünschte Dimensionalität erhält. Um die Netzwerkausgaben in den richtigen Wertebereich zu überführen, werden diese von einer Softmaxfunktion aktiviert, bevor die Fehlerfunktion berechnet wird:

$$\text{Softmax}(\hat{Y'})_c = \frac{e^{\hat{Y}'_c}}{\sum_{j \in C} e^{\hat{Y}'_j}} \forall c \in C \tag{3.2}$$

Hierbei ist $\hat{Y'}$ ein Ausgabevektor des Netzwerks und $\hat{Y}'_c$ der Eintrag des Vektors welcher zur Klasse c zugeordnet ist. Der resultierende Vektor kann als Wahrscheinlichkeitsverteilung über die unterschiedlichen Klassen im Bezug zur Eingabe interpretiert werden. Mit den vorgenommenen Modifikationen kann das ResNet Modell auf dem Trainingsdatensatz optimiert werden. Die für das Fine-Tuning und Attention-Training verwendeten Hyperparamter werden im Experimentalteil in Abschnitt **??** auf Seite **??** erläutert.

3.4 Attention Training

Delf erzeugt eine große Anzahl an lokalen Deskriptoren, um Bilder zu beschreiben. Da jeder Deskriptor nur einen Ausschnitt des Originalbildes beschreibt, werden auch Deskriptoren für wenig aussagekräftige Bereiche, wie z.B. Teile des Himmels erstellt. Diese Deskriptoren beanspruchen nicht nur zusätzliche Rechenzeit während des Matchingprozesses sondern können auch zu falsch positiven Matches führen. Ziel der zweiten Trainingsphase ist es daher auf Basis dieser Deskriptoren ein Netzwerk zu trainieren, welches in der Lage ist die Qualität der Deskriptoren zu bewerten und so ungeeignete Kandidaten herauszufiltern. Zur Erstellung der Deskriptoren dient das ResNet-50, welches in der ersten Phase trainiert wurde. Als Deskriptoren werden dabei die Ausgaben aus dem dritten ResNet-Blocks genutzt (vgl. Abb. 3.2b). Die Ausgaben haben eine Dimensionalität von $w \times h \times 1024$, wobei w und h abhängig von der Breite und

Höhe des Eingabebildes sind. Die einzelnen Koordinaten der 1024 Featuremaps lassen sich jeweils auf einen Bildbereich in der Eingabe zurückführen. So kann die Ausgabe in $w \times h$ Deskriptoren der Größe 1024 eingeteilt werden. Wie auch beim Fine-Tuning werden die Bilder für das Training zunächst quadratisch zugeschnitten. Anschließend werden sie auf eine zufällige Seitenlänge zwischen 255 bis 720 Pixel skaliert. Die Skalierung hat Einfluss auf die Beschaffenheit der entstehenden Deskriptoren. Durch das zufällige Skalieren der Trainingsbilder lernt das Attention-Netzwerk mit Deskriptoren aus verschiedenen Skalen umzugehen. Während dem Attention-Training werden die Parameter des ResNets nicht mehr verändert. Die Schichten nach dem Extraktionspunkt in Block 3 werden nicht mehr benötigt und können verworfen werden.

Aufgabe des Attention-Netzwerks ist es für eine Eingabe an Deskriptoren der Form $w \times h \times 1024$ eine

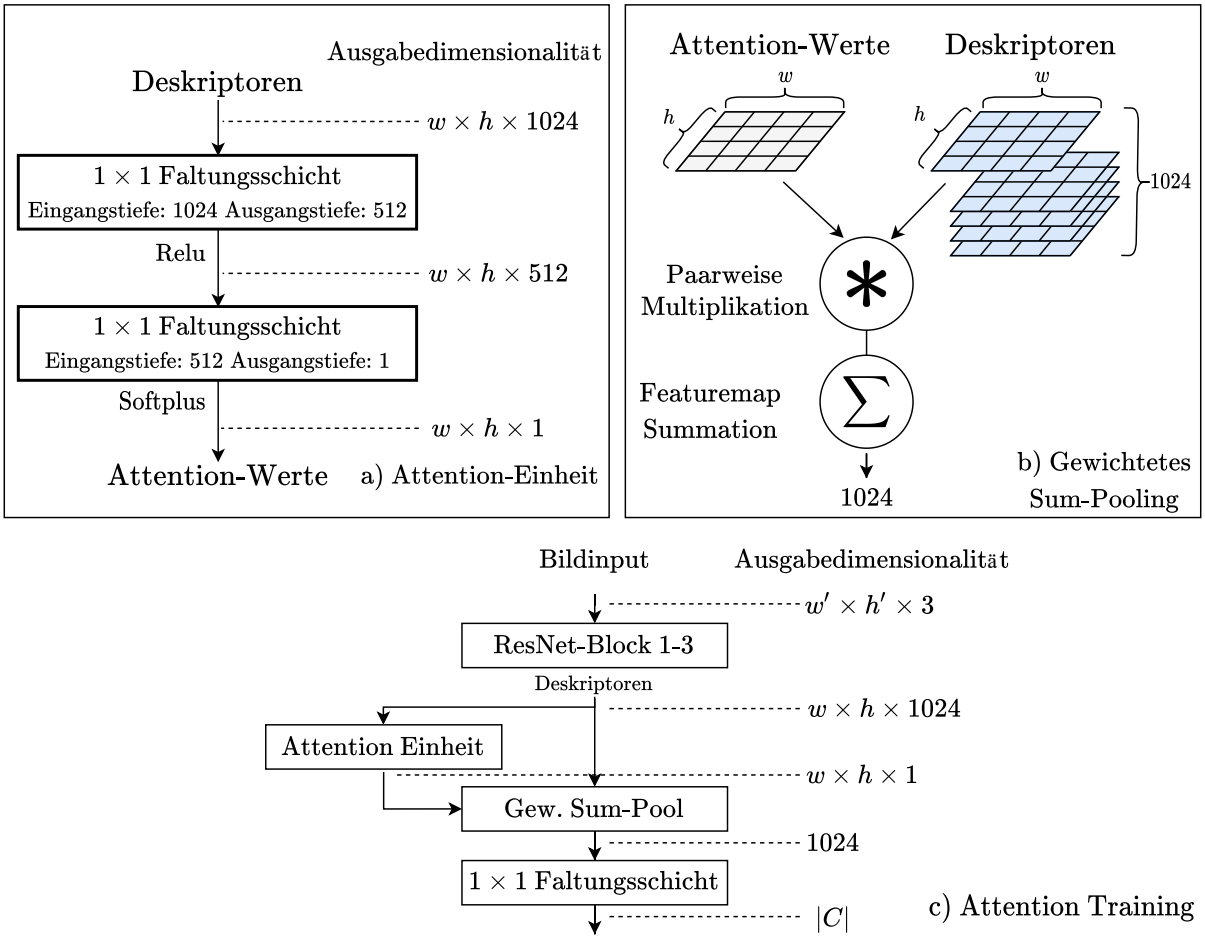


Abbildung 3.3: Architektur des Attention Trainings

einzelne Featuremap der Größe $w \times h$ zu generieren, dessen Werte jeweils einen Deskriptor bewerten. Wichtig ist dabei, dass die Berechnung der einzelnen Attention-Werte nur von den Werte der dazugehörigen Deskriptoren abhängen dürfen. Dies kann durch faltenden Schichten mit 1×1-Filtermasken realisiert werden. Die Attention-Einheit besteht aus zwei solcher Schichten, welche die Merkmalstiefe der Deskriptoren sukzessive auf 1 reduzieren (vgl. Abb. 3.3a). Um die Parameter der Attention-Einheit zu optimieren müssen ihre Ausgaben zur Lösung der Trainingsaufgabe beitragen. Da die Attention-Werte später genutzt werden um zu entscheiden, welche Deskriptoren Einfluss auf die Lösung der Retrieval-

aufgabe haben ist es sinnvoll sie auch beim Training in einer Form zu nutzen, die den Einfluss der Deskriptoren zur Lösung der Klassifikationsaufgabe reguliert. Die Attention-Werte werden zur Gewichtung der Deskriptoren genutzt und dafür elementweise mit den Featuremaps der Deskriptoren Multipliziert. Anschließend werden die gewichteten Featuremaps zu jeweils einem Wert summiert. Als Ausgabe ergibt sich ein 1024 dimensionaler Vektor (vgl. Abb. 3.3b). Abschließend muss die Ausgabe in eine passende Form für die Berechnung der Kreuzentropie gebracht werden. Dies geschieht analog wie im Fine-Tuning durch Verwendung einer 1×1-Faltungsschicht und anschließender Softmaxaktivierung der Ausgabe (vgl. 3.3c).

3.5 Extraktion und Verarbeitung

Nachdem das Training der Modelle abgeschlossen ist, kann mit der Extraktion aller benötigten Informationen über den Retrievaldatensatz begonnen werden. Netzwerkparameter werden ab jetzt nicht mehr modifiziert. Schichten und Operationen nach der Attention-Einheit erfüllen daher keine Zweck mehr und können entfernt werden.

3.5.1 Multi-Skalen-Extraktion

Für jedes Bild des Retrievaldatensatzes werden die lokalen Deskriptoren am Extraktionspunkt nach dem dritten ResNet-Block und die dazugehörigen Attention-Werte nach der Attention-Einheit extrahiert. Da die Skalierung des Bildinhalts Einfluss auf die resultierenden Deskriptoren hat, wird für jedes Bild eine Reihen von unterschiedlich skalierten Versionen betrachtet. Dies ist vergleichbar mit der Verwendung des Scale Spaces im SIFT-Verfahren (vgl. Kap.2 Abs.2) und soll zur Invarianz gegenüber Skalierungsoperationen beitragen. Für jedes Bild werden sechs unterschiedlichen Skalen mit Skalierungsfaktoren zwischen 2 und $\frac{1}{4}$ erstellt, wobei sich benachbarte Skalen um den Faktor $\sqrt{2}$ unterscheiden.

3.5.2 Deskriptorlokalisierung

Für den weiteren Verlauf des Verfahrens muss jeder lokale Deskriptor einem Bereich des Eingangsbildes zuordenbar sein. Bei allen verwendeten Schichten des Modells bis zum Extraktionspunkt handelt es sich um Faltungs- oder Poolingschichten. Von welchen Bereichen der Eingabe die Ausgaben dieser Schichten abhängen lässt sich an drei Parametern festmachen. Die Größe der Filtermasken k bestimmt die Größe des Einflussbereiches einzelner Ausgaben. Die Verschiebung der Filtermasken bzw. die Schrittgröße s bestimmt die Verschiebung zwischen den Einflussbereichen benachbarter Ausgaben. Das Padding p bestimmt die Größe des Pufferbereichs, der der Eingabe hinzugefügt wird, und sorgt so für eine initiale Verschiebung der Einflussbereiche. Das Padding ist im folgenden immer symmetrisch und wird daher an jeder Seite der Eingabe hinzugefügt. Betrachtet man mehrere aufeinanderfolgende Schichten, so lassen

sich die Einflüsse dieser Parameter wie folgt rekursiv berechnen:

$$\hat{k}_n = \hat{k}_{n-1} + ((k_n - 1) * \hat{s}_{n-1}) \tag{3.3}$$

$$\hat{s}_n = \hat{s}_{n-1} * s_n \tag{3.4}$$

$$\hat{p}_n = \hat{p}_{n-1} + (p_n * \hat{s}_{n-1}) \tag{3.5}$$

$$\hat{k}_0 = k_0 \tag{3.6}$$

$$\hat{s}_0 = s_0 \tag{3.7}$$

$$\hat{p}_0 = p_0 \tag{3.8}$$

Wobei $\hat{k}_n$, $\hat{s}_n$ und $\hat{p}_n$ die Größe der Einflussbereiche, Schrittgröße und Paddinggröße im Bezug zur ursprüngliche Eingabe nach n Schichten repräsentieren. k_n, s_n und p_n zeigen die selben Größen für Schicht n im Bezug zur Ausgabe der vorangehenden Schicht. In Abbildung 3.4 werden diese Berechnungen exemplarisch erklärt. Berechnet man diese Werte für den Extraktionspunkt nach dem dritten ResNet-Block

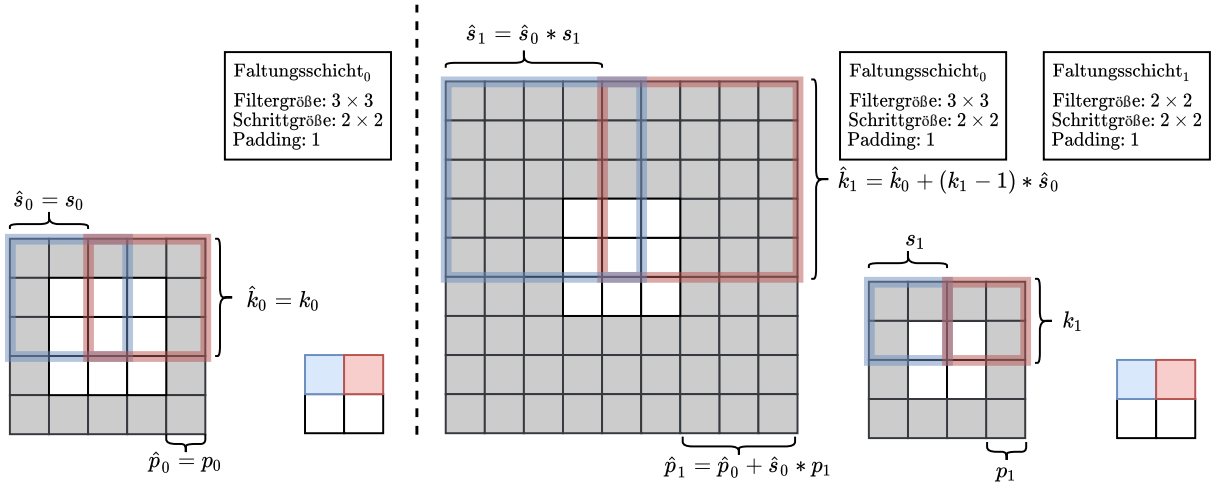


Abbildung 3.4: Berechnung des Einflussbereichs einzelner Ausgaben in der ursprünglichen Eingabe. Links nach einer Faltungsschicht, Rechts nach zwei Schichten. Graue Bereiche repräsentiert hinzugefügten Puffer.

ergibt sich für jeden lokalen Deskriptor ein quadratischer Einflussbereich $\mathbf{k}$ mit einer Seitenlänge von 267 Pixeln im Ursprungsbild. Die Verschiebung zwischen Einflussbereichen benachbarter Deskriptoren $\mathbf{s}$ beträgt dabei 16 Pixel. Das Ursprungbild erhält bis zu dieser Schicht ein effektives Padding $\mathbf{p}$ von 133 Pixeln in jede Richtung. Aufgrund dieser Werte lassen sich die Einflussbereiche für alle $w \times h$ Deskriptoren, die am Extraktionspunkt anfallen wie folgt berechnen:

$$x_{min}(i,j) = i * \mathbf{s} - \mathbf{p} \tag{3.9}$$

$$x_{max}(i,j) = x_{min}(i,j) + \mathbf{k} \tag{3.10}$$

$$y_{min}(i,j) = j * \mathbf{s} - \mathbf{p} \tag{3.11}$$

$$y_{max}(i,j) = y_{min}(i,j) + \mathbf{k} \qquad \text{wobei } 0 \leq i < w \text{ und } 0 \leq j < h \tag{3.12}$$

3.5.3 Deskriptorselektion

Im nächsten Verarbeitungsschritt werden die aussagekräftigsten lokalen Deskriptoren auf Basis der Attention-Werte selektiert. Da während der Deskriptorextraktion unterschiedliche skalierte Bildversionen betrachtet wurden steht vor der Selektion ein große Anzahl an Deskriptoren zur Auswahl, die teilweise auch stark überlappende Bildbereiche beschreiben. Das kann dazu führen, dass aus einigen Bildbereichen sehr viele Deskriptoren ausgewählt werden, die zwar individuell viel Information über das Bild bereitstellen, in der Summe aber wenig Nutzen haben, da der gleiche Bildbereich vielfach beschrieben wird. Da die Anzahl der selektierten Deskriptoren pro Bild begrenzt ist, kann dies zu einer Verdrängung von Deskriptoren aus anderen wichtigen Bildbereichen führen. Um dem entgegenzuwirken werden zunächst Deskriptoren, die stark überlappende Bildbereiche beschreiben mittels Non-Maximum-Suppression aussortiert. Als Metrik für die Überlappung wird hierfür die Intersection-over-Union(IoU) zwischen den Einflussbereichen der Deskriptoren berechnet. Also das Verhältnis zwischen überlappenden Flächeninhalt zu gemeinsam eingenommenem Flächeninhalt der Einflussbereiche.

$$\mathrm{IoU}(e_1, e_2) = \frac{e_1 \cap e_2}{e_1 \cup e_2} \tag{3.13}$$

Wobei e_1 und e_2 Einflussbereiche zweier Deskriptoren sind. Der Non-Maximum-Suppression Algorithmus (siehe Alg. 3.1) erhält als Eingabe eine Liste mit den Einflussbereichen der lokalen Deskriptoren $\mathcal{E}$, die dazugehörigen Attention-Werte $\mathcal{A}$, sowie einen Schwellwert T der maximal tolerierten IoU zwischen den Einflussbereichen der Deskriptoren. Zunächst werden die Kandidaten nach Attention-Wert sortiert. Der Einflussbereich mit dem höchsten Attention-Wert wird ausgewählt und von der Kandidatenliste in die Ergebnisliste geschoben. Anschließend wir die IoU zwischen dem ausgewählten Einflussbereich und den übrigen Einflussbereichen der Kandidatenliste berechnet. Kandidaten dessen IoU den Schwellwert T überschreiten werden aus der Kandidatenliste entfernt. Anschließend wird aus der Kandidatenliste erneut der Einflussbereich mit dem nun höchsten Attention-Wert ausgewählt und der Prozess wiederholt, bis die Kandidatenliste leer ist. Abschließend wird die Ergebnisliste mit den ausgewählten Bereichen und dazugehörigen Attention-Werten zurückgegeben.

Algorithmus 3.1: Non-Maximum-Suppression

Input: $\mathcal{E} \leftarrow \{e_1, \ldots, e_n\}, \mathcal{A} \leftarrow \{a_1, \ldots, a_n\}, T$
$\mathcal{S} \leftarrow \emptyset$
while $\mathcal{E} \neq \emptyset$ **do**
  $x \leftarrow \mathrm{argmax}(\mathcal{A})$ `// Wähle bestbewertete Box`
  $currBox \leftarrow e_x$
  $\mathcal{E} \leftarrow \mathcal{E} \setminus \{e_x\}$ `// Verschiebe Box von Kandidaten in Ergebnisliste`
  $\mathcal{A} \leftarrow \mathcal{A} \setminus \{a_x\}$
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{(e_x, a_x)\}$
  foreach $e_i \in \mathcal{E}$ **do**
    `// Entferne Boxen mit IoU > T zu currBox aus Kandidatenliste`
    if $IoU(e_i, currBox) > T$ **then**
      $\mathcal{E} \leftarrow \mathcal{E} \setminus \{e_i\}$
      $\mathcal{A} \leftarrow \mathcal{A} \setminus \{a_i\}$
return $\mathcal{S}$

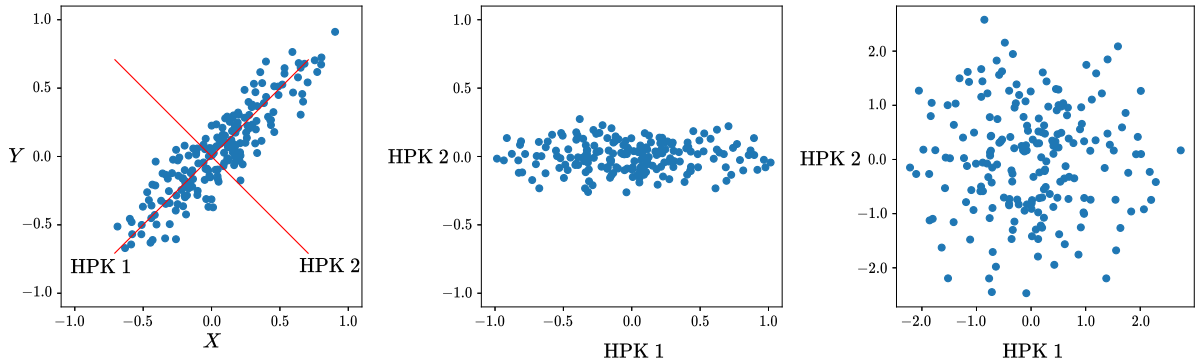
Im Delf-Verfahren wird für die Vorsortierung ein IoU-Schwellwert von 0.8 genutzt. Nach der Vorsortierung findet die finale Selektion der Deskriptoren statt. Für jedes Bild werden hierbei von den verbliebenen Deskriptoren die 1000 Deskriptoren mit den höchsten Attention-Werten ausgewählt.

3.5.4 Datentransformation und Dimensionsreduktion

Ziel des letzten Vorverarbeitungsschritts ist es die Daten der lokalen Deskriptoren in eine Form zu bringen, die ein effizientes Vergleichen möglich macht. Obwohl durch den Selektionsprozess bereits die Anzahl an lokalen Deskriptoren je Bild drastisch reduziert wurde ist die aktuelle Repräsentation der einzelnen Bilder noch sehr groß. Mit 1000 Deskriptoren mit je 1024 Dimensionen umfasst die Beschreibung jedes Bildes über eine Millionen Werte. Durch die folgenden Transformationen, wird die Größe der lokalen Deskriptoren und damit der gesamt Repräsentation stark reduziert. Die für Delf beschriebenen Transformationen bezeichnen die Autoren dabei als "common practice"(vgl. [2] Kap.4.3) und beziehen sich dabei auf das von Jégou und Chum in [28] untersuchte Vorgehen zur Transformation von VLAD und BOW-Deskriptoren. Tatsächlich werden Deskriptoren auch in anderen Image Retrieval Verfahren [14] [16] auf ähnliche Weise verarbeitet.
Zunächst werden die Deskriptoren der Länge nach normiert. Anschließend wird auf ihnen eine Hauptkomponentenanalyse durchgeführt. Die resultierende Transformationsmatrix soll dabei repräsentative für die Deskriptoren des gesamten Datensatzes sein, daher werden bei der Analyse Deskriptoren des ganzen Datensatzes oder zumindest eines erheblichen Teils betrachtet. Ziel der Hauptkomponentenanalyse ist es die Daten anhand neuer Dimensionen, den sogenannten Hauptkomponenten, darzustellen. Die Richtungen der Hauptkomponenten ergeben sich aus einer Linearkombination der bisherigen Dimensionen und sind so gewählt, dass für die transformierten Daten keine Korrelation zwischen den Dimensionen besteht. Eine weitere Eigenschaft dieser Darstellung ist, dass die einzelnen Hauptkomponenten jeweils den größtmöglichen Anteil, der in den Daten vorhandenen Varianz abbilden. D.h. die Hauptkomponente, die den größten Teil der Varianz abbildet ist die Richtung entlang welcher die Daten am stärksten Variieren. Die Hauptkomponente, die den nächst größten Anteil der Varianz erklärt beschreibt den größtmöglichen Anteil der verbleibenden Varianz. Dies führt dazu, dass der wesentliche Teil der Varianz von wenigen Dimensionen erklärt wird. Dadurch ist es möglich einen großen Teil der Dimensionen zu entfernen, ohne einen starken Informationsverlust zu erleiden.
Sei $X_{n\times d}$ eine Matrix von n zu analysierenden Deskriptoren mit je d Dimensionen. Zunächst werden die Daten zentriert, sodass ihr Mittelwert entlang der d Dimensionen 0 ist:

$$x'_{i,j} = x_{i,j} - \frac{1}{n}\sum_{k=1}^{n} x_{k,j} \qquad \text{für } 0 < i \leq n, 0 < j \leq d \tag{3.14}$$

Häufig werden die Daten anschließend durch die Standardabweichung der einzelnen Dimensionen geteilt um die Varianz je Dimension auf 1 zu setzten. Je größer die Varianz in den Dimensionen ist, desto größer ihr Anteil in den berechneten Hauptkomponenten. Diese Standardisierung wird daher durchgeführt, wenn die Größe der Varianzen der ursprünglichen Dimensionen nicht adäquat ihre Bedeutsamkeit widerspiegeln. Für Delf wird keine Anpassung der Varianz durchgeführt.
Nach der Hauptkomponentenanalyse werden die Daten in den Raum der Hauptkomponenten transformiert. Die neuen Dimensionen sind dabei nach der von ihnen erklärten Varianz sortiert (vgl. Abb. 3.5ab).



a) Berechnung der Hauptkomponenten b) Transformation c) Transformation + Weißen

Abbildung 3.5: Beispiel zur Hauptkomponentenanalyse auf 2D-Daten

Delf erhält von den transformierten Deskriptoren nur die ersten 40 Dimensionen und erzeugt damit eine deutlich kompaktere Repräsentation der Bilder. Wie sich die Anzahl erhaltener Dimensionen auf das Retrievalergebnis auswirkt wird in Kapitel **??** auf Seite **??** experimentell untersucht. Weiterhin sieht das Verfahren vor die Deskriptorendaten zu weißen, d.h. in eine Form zu bringen, in der zwischen den Dimensionen keine Korrelation herrscht und die Varianz entlang der Dimensionen jeweils 1 beträgt (vgl. Abb. 3.5c). Die erste Voraussetzung ist dabei durch die Transformation in den Raum der Hauptkomponenten automatisch erfüllt. Für die Anpassung der Varianz müssen die Daten durch die Standardabweichungen der neuen Dimensionen geteilt werden. Bevor die Deskriptoren für das Matching genutzt werden, werden sie erneut der Länge nach normiert.

3.6 Matching

Sobald die Daten der lokalen Deskriptoren des Datensatzes erhoben, verarbeitet und in den Bildern lokalisiert sind ist das Delf-System in der Lage Suchanfragen zu bearbeiten. Wird ein Bild als Anfrage an das System gestellt werden auch für dieses Bild alle notwendigen Informationen berechnet, wie im letzten Abschnitt beschrieben. Um die Anfrage zu beantworten müssen die Bilder im Suchdatensatz anhand ihrer Ähnlichkeit zum Anfragebild sortiert werden. Hierfür wird jede mögliche Kombination an Bildpaaren aus der Anfrage und dem Suchdatensatz einzeln untersucht. Zunächst wird dabei ein initiales Matching zwischen den Deskriptoren des Anfragebildes $D_{\mathcal{A}}$ und den Deskriptoren des aktuell betrachteten Bildes $D_{\mathcal{S}}$ aus dem Suchdatensatz hergestellt. Für jeden lokalen Deskriptor $d_{\mathcal{A}i}$ wird dabei der ähnlichste Deskriptor in $D_{\mathcal{S}}$ gesucht[6]. Als Metrik wird hierfür die euklidische Distanz berechnet. Ist die Distanz zwischen $d_{\mathcal{A}i}$ und dem ähnlichsten Deskriptor in $D_{\mathcal{S}}$ kleiner als ein Schwellwert T werden die Deskriptoren als potentielles Match vermerkt (vgl. Alg. 3.2). Delf nutzt für T standardmäßig einen Wert vom 0.8.

Anschließend wird überprüft, ob sich die gefundenen Matches auch geometrisch erklären lassen. Die Annahme dabei ist, dass es eine affine Transformation vom Anfragebild auf das betrachtete Bild der

[6] In der offiziellen Implementierung wird für die effiziente Suche der ähnlichsten Deskriptoren ein k-d-Baum [29] über die Deskriptoren des Anfragebildes erstellt. Eine Effizienzsteigerung mit k-d-Bäumen lässt sich jedoch nur erzielen, solang ein angemessenes Verhältnis zwischen der Anzahl der Datenpunkte n und Dimensionalität d der Daten besteht ($n >> 2^d$)[30], was in diesem Anwendungsfall nicht gegeben ist.

Algorithmus 3.2: Initiales Deskriptor Matching

Input: $D_{\mathcal{A}} \leftarrow \{d_{\mathcal{A}1}, \ldots, d_{\mathcal{A}n}\}$, $D_{\mathcal{S}} \leftarrow \{d_{\mathcal{S}1}, \ldots, d_{\mathcal{S}n}\}$, T
$\mathcal{M} \leftarrow \emptyset$
foreach $d_{\mathcal{A}i} \in D_{\mathcal{A}}$ **do** `// Für jeden Deskriptor in` $\mathcal{A}$
  $closestDistance \leftarrow \inf$
  $closestDescriptor \leftarrow -1$
  foreach $d_{\mathcal{S}j} \in D_{\mathcal{S}}$ **do** `// Finde ähnlichsten Deskriptor in` $\mathcal{S}$
    if $||d_{\mathcal{A}i} - d_{\mathcal{S}j}||_2 < closestDistance$ **then**
      $closestDistance \leftarrow ||d_{\mathcal{A}i}, d_{\mathcal{S}j}||_2$
      $closestDescriptor \leftarrow j$
  if $closestDistance < T$ **then** `// Falls näher als` T
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{(i, closestDescriptor)\}$ `// Füge potentielles Match hinzu`
return $\mathcal{S}$

Suchdatenbank geben muss, welche die Deskriptorpositionen im Anfragebild auf die Positionen der gefundenen Matches im Suchbild projiziert, falls das betrachtete Bildpaar tatsächlich den gleichen Bildinhalt zeigt.

Für die geometrische Verifikation müssen die Deskriptoren der Matches Koordinaten in ihren dazugehörigen Bildern zugeordnet werden. Hierfür werden die zuvor berechneten Einflussbereiche der Deskriptoren (siehe Kap. 3.5.2 auf Seite 14) genutzt. Die Zentren der Einflussbereiche dienen dabei als Referenzpunkte für die Deskirptoren. Für die Verifikation wird der RANSAC-Algorithmus (RANdom SAmple Consesus) [31] von Fischler und Bolles genutzt. Als Eingabe erhält der Algorithmus eine Liste mit den Positionen der Deskriptoren aus den potentiellen Matches $\mathcal{L}$, einen Schwellwert T_{in}, der festlegt, wie weit die projizierte Positionen maximal von den tatsächlichen Postionen ihres Matches entfernt sein dürfen, um noch von einer Transformation erklärt zu werden und die Anzahl an Versuchen $numTrials$, die unternommen werden um eine passende Transformation zu finden.

Zu Beginn werden zufällig Matchpaare aus $\mathcal{L}$ gewählt, mit denen sich eine affine Transformation zwischen den Bildern bestimmen lässt. Für eine affine Ebenentransformation benötigt man drei Koordinatenpaare, um eine Transformationsmatix bestimmen zu können. Nun wird überprüft welche Matches sich von dieser Transformation erklären lassen. Sind die Abstände der transformierten Deskriptorpositionen des Anfragebildes zu ihren Matches im aktuellen Suchbild innerhalb des Schwellwerts T_{in} werden sie als Inlier bezeichnet. D.h. sie können von der Transformation erklärt werden. Dieser Prozess wird mehrfach wiederholt, wobei jedes Mal zufällige Matchpaare für die Berechnung der Transformationsmatrizen genutzt werden. Nach $numTrails$ Versuchen gibt der Algorithmus die Transformation mit den dazugehörigen Inliern zurück, welche die meisten Matches erklären könnte (vgl. Alg. 3.3). Für Delf werden standardmäßig 1000 RANSAC-Versuche geeignete Transformationen zu finden. Die Schwellwertdistanz T_{in} beträgt dabei 20.

Delf nutzt die Anzahl erklärter Deskriptormatches als Metrik für die Ähnlichkeit zwischen Bildpaaren. So können die Bilder der Suchdatenbank anhand ihrer Ähnlichkeit sortiert und dem Nutzer zurückgegeben werden. In Kapitel **??** auf Seite **??** werden Parameter und Metriken zur Bestimmung von Ähnlichkeiten zwischen Bilder experimentell betrachtet.

Algorithmus 3.3: RANSAC

Input: $\mathcal{L} \leftarrow \{(l_{\mathcal{A}1}, l_{\mathcal{S}1}), \ldots, (l_{\mathcal{A}k}, l_{\mathcal{S}k})\}, T_{in}, numTrials$
$bestModel \leftarrow$ None
$bestInliers \leftarrow \emptyset$
for $i = 0, i < numTrials, i$++ **do**
    `// Berechne affine Transformation aus zufälligen Matchpaaren`
    $inliers \leftarrow \emptyset$
    $selectedMatches \leftarrow$ drawRandomMatches$(\mathcal{L}, pairsNeeded = 3)$
    $model \leftarrow$ calcModel$(selctedMatches)$
    foreach $(l_{\mathcal{A}j}, l_{\mathcal{S}j}) \in \mathcal{L}$ **do**
        `// Sammle von Transformation erklärte Paare`
        $transformedLocation \leftarrow$ transform$(l_{\mathcal{A}j}, model)$
        if $||l_{\mathcal{S}j} - transformedLocation||_2 < T_{in}$ **then**
            $inliers \leftarrow inliers \cup \{(l_{\mathcal{A}j}, l_{\mathcal{S}j})\}$
    if *len*$(inliers) >$ *len*$(bestInliers)$ **then**
        $bestInliers \leftarrow inliers$
        $bestModel \leftarrow model$
return $bestInliers, bestModel$ `// Gib bestes Modell + Inlier zurück`

3.7 Verfahrensunterschiede im Delf Artikel

Die Verfahrensbeschreibung aus dem zu Delf erschienen Artikel (vgl. [2] Kap.4 und Kap 5.1) weist einige Unterschiede zu dem dazugehörigen veröffentlichten Quellcode auf. Teilweise sind diese Unterschiede der Kürze des Artikels geschuldet. Verfahrensschritte werden meist nur oberflächlich beschrieben, was einen gewissen Interpretationsspielraum für die tatsächliche Umsetzung zulässt. Einige Schritte, wie beispielsweise die Verwendung von Non-Maximum-Suppression zur Vorsortierung bei der Auswahl der lokalen Deskriptoren (siehe Kap. 3.5.3 auf Seite 16), die für das generelle Verständnis des Verfahrens nicht essentiell sind, werden im Artikel nicht erwähnt.
Wesentliche Unterschiede gibt es in der Strukturierung der extrahierten Deskriptoren des Suchdatensatzes, sowie in der Verarbeitung von Suchanfragen. Das im Artikel beschriebene Vorgehen nutzt dabei einen invertierten Index in Kombination mit Produkt Quantisierung [32], um lokale Deskriptoren effizient zu matchen. Dieser Ansatz ist insbesondere für die Suche auf sehr großen Datensätzen besser geeignet, als das Vorgehen im veröffentlichten Quellcode. Für die vergleichsweise überschaubaren historischen Datensätze, mit denen sich diese Arbeit beschäftigt stellt dies jedoch kein Problem dar.
Im Artikel werden beim beantworten einer Anfrage jeweils nicht nur die Deskriptoren eines Bildpaares, sonder alle Deskriptoren des Suchdatensatzes betrachtet. Zunächst wird ein invertierter Index mit einem Codebuch von 8k verschiedenen Visuellen Wörtern über den lokalen Deskriptoren erstellt. Das heißt die lokalen Deskriptoren werden mit Hilfe des K-Means-Algorithmus[23] jeweils einem der 8k Clusterzentren zugeordnet. In der zum Cluster gehörigen Postingliste wird dann ein neuer Eintrag erstellt, der Informationen über die Beschaffenheit des Deskriptors und des Bildes, aus dem er extrahiert wurde enthält. Hierfür wird das Residuum $r(x)$ zwischen dem betrachteten Deskriptor x und dem ihm zugeordneten Cluster-Zentrum $q(x)$ berechnet.

$$r(x) = x - q(x) \tag{3.15}$$

Der 40-dimensionale Residuenvektor wird anschließend mit Hilfe eines Produkt Quantisierers auf einen 50 bit Code abgebildet. Der Vektor wird dafür in m Subvektoren aufgeteilt. Die Subvektoren werden dann mittels K-Means-Algorithmus einem von k Clusterzentren zugeordnet. Der Code ergibt sich aus den konkatenierten Indizes der zugeordneten Clusterzentren. Im Fall von Delf werden die Residuenvektoren in 10 Subvektoren zerlegt, die jeweils einem von 2^5 Clusterzentren zugeordnet werden. Der erzeugte Code bildet die Information über den Deskriptor, welche in den Postinglisten hinterlegt wird. Bei der Verarbeitung einer Anfrage werden die lokalen Deskriptoren der Anfrage zunächst wieder quantisiert und Postinglisten zugeordnet. Anschließend werden ihre Residuenvektoren mit den Einträgen der Postinglisten verglichen. Hierfür werden die 50 bit Codes wieder über ihre zugehörigen Clusterzentren repräsentiert und die Euklidische Distanz der Zentren zum Residuenvektor des Anfragedeskriptors berechnet (vgl. Abb. 3.6). Auf diese Weise findet das System für jeden lokalen Anfragedeskriptor die nächsten 60 Deskriptoren aus dem Suchdatensatz. Diese Matches werden pro Bild akkumuliert. Anschließend findet eine geometrische Verifikation der Matches mit Hilfe des RANSAC-Algorithmus statt. Die Anzahl an verifizierten Matches bilden auch hier die Grundlage für die Bewertung der Ähnlichkeit zwischen Bildern.

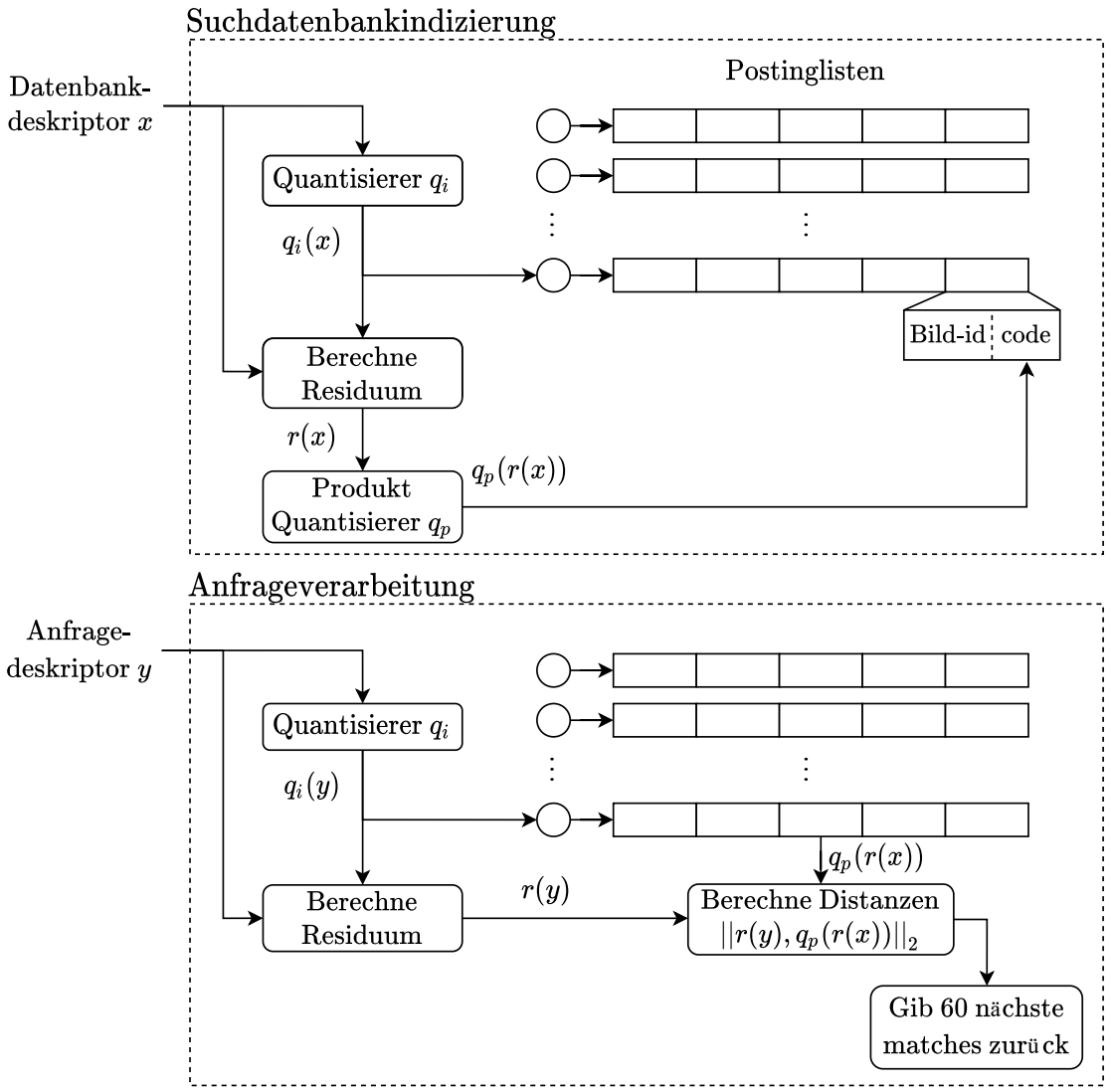


Abbildung 3.6: Invertierter Index und Produkt Quantisierung zur Anfrageverarbeitung (vgl. [32] Fig.5)

4 Parameter Analyse

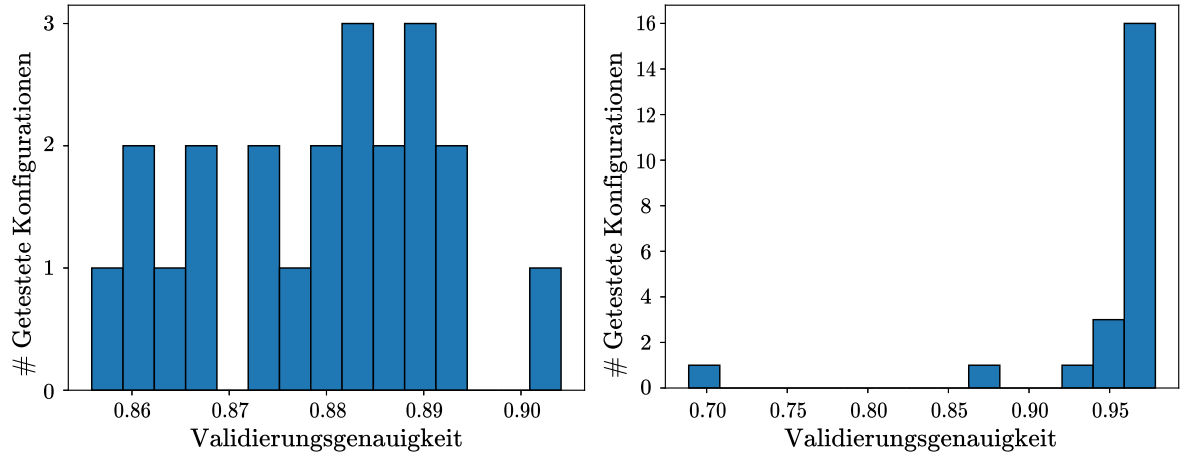
Ein wesentliches Ziel dieser Arbeit ist es das Delf-Verfahren insbesondere für den Anwendungsfall des Retrievals von historischen Abbildungen zu optimieren. Hierfür werden ein Reihen an Parametern entlang der Delf-Pipeline variiert und ihre Einflüsse, auf die Retrievalergebnisse analysiert. Die benötigte Rechenleistung zur Durchführung der Experimente wird freundlicherweise vom Zentrum für Informationsdienste und Hochleistungsrechnen (ZIH) an der TU Dresden zur Verfügung gestellt. Das verwendete HPC-DA System[1] ermöglicht es viele Experimente parallel und effizient durchzuführen, sodass auch im zeitlich überschaubaren Rahmen dieser Arbeit eine große Anzahl unterschiedlicher Konfigurationen getestet werden können.

4.1 Hyperparameteroptimierung der Trainingsphasen

Die ersten Experimentalreihen befassen sich mit den Trainingsphasen des Delf-Verfahrens. Um bei späteren Retrievalversuchen gute Ergebnisse erzielen zu können, benötigt man Modelle die in der Lage sind aussagekräftige Bildrepräsentationen zu erstellen. Für die Experimente zum Modelltraining wird dabei angenommen, dass die Güte, der von einem Modell erzeugten Deskriptoren bzw. der von einem Modell getroffenen Auswahl an Deskriptoren positiv mit der Fähigkeit, der Modelle korreliert die beim Training gestellte Klassifikationsaufgaben zu lösen. Um die Modelle zu bewerten wird daher der Fehler, in Form der Kreuzentropie betrachtet, der auftritt wenn das Modell einen Validierungsdatensatz klassifiziert. Vor Beginn jedes Experiments werden zufällig 20% der Trainingsbilder (siehe Kap. 3.2 auf Seite 11) für die Validierung ausgewählt. Die Validierungsdaten stehen dem Modell während des Trainingsprozesses nicht zur Verfügung. Daher können sie genutzt werden, um zu überprüfen, ob ein Modell auch auf ungesehenen Daten vergleichbare Ergebnisse erzielt.
Der Erfolg des Trainingsprozesses hängt wesentlich von der zu trainierenden Architektur und der vorhandenen Datenlage ab. Einfluss haben außerdem Parameter, die den Ablauf des Trainingsprozesses beeinflussen. Die Experiments, die in diesem Abschnitt besprochen werden befassen sich mit der Suche nach optimalen Werten für drei dieser sogenannten Hyperparameter. Betrachtet wird die Anzahl der Trainingsepochen, die Lernrate, die bestimmt wie stark Netzwerkparameter in einem Optimierungsschritt angepasst werden, sowie der Faktor γ mit dem die Lernrate alle 10 Epochen multipliziert wird[2]. Die übrigen Hyperparameter sind für alle Experimente fest gesetzt. Die Modelle werden mit einer Batchgröße von 8 trainiert und mittels Stochastic Gradient Decent (kurz SGD [3]) optimiert. Für die zu untersuchenden Hyperparameter werden jeweils Wertebereiche definiert. Die Länge des Trainings kann zwischen 10 und 40 Epochen dauern. Die initiale Lernrate wird zwischen 0.01 und 0.001 gewählt und der γ-Faktor liegt im Bereich zwischen 1 und 0.1.

[1] `https://tu-dresden.de/zih/hochleistungsrechnen/hpc`, zuletzt besucht am 28.07.20
[2] Als weiterer Hyperparameter wurde weight decay untersucht, allerdings lies sich hier kein signifikanter Einfluss auf den Validierungsfehler feststellen. Die Ergebnisse hierzu finden sich im Anhang ab Seite **??**
[3] `https://pytorch.org/docs/stable/optim.html#torch.optim.SGD`, zuletzt besucht am 30.07.20

Um den Suchraum effizient erkunden zu können wird das von Norman Koch entwickelte NNOPT-Tool verwendet, um neue Hyperparameterkonfigurationen zu erstellen und auf dem HPC-System zu testen. Das NNOPT-Tool, das auf dem Hyperopt-Paket [33] von Bergstra, Yamins und Cox basiert, wählt automatisch Werte für die zu untersuchenden Hyperparameter innerhalb der definierten Wertebereiche aus und startet mit diesen Experimente auf dem Großrechner. Ist ein Experiment abgeschlossen erhält NNOPT zur Bewertung der Konfiguration den erzielten Validierungsfehler. Neue Konfigurationen werden bevorzugt in der Nähe von bereits getesteten Konfigurationen erstellt, die gute Ergebnisse erzielt haben. Dies erlaubt es schneller optimale Hyperparameterwerte zu finden, als mit einer zufälliger Suche. Für das Fine-Tuning wurden auf diese Art 22 unterschiedliche Konfigurationen getestet. Es lässt sich beobachten, dass fast alle getesteten Konfigurationen Modell erzeugen, die in der Lage sind die Trainingsaufgabe sehr gut zu lösen. Im Schnitt erzielen die trainierten Modelle eine Klassifikationgenauigkeit von 95.13% auf den Validierungsdaten[4], wobei 20 der 22 Modelle eine Genauigkeit von über 90% erreichen(vgl. Abb.4.1b). Da auf Grund der guten Ergebnisse nur wenig Potential für Verbesserung besteht und die beobachtet Varianz der Ergebnisse unterschiedlicher Konfigurationen gering ist, werden keine weiteren Testläufe zur Hyperparameteroptimierung durchgeführt. Es sei jedoch erwähnt, dass sich auf Grund der im Verhältnis zu Größe der Suchraumes geringen Anzahl an Testläufen keine eindeutigen Abhängigkeiten zwischen Hyperparametern und Klassifikationsperformanz ableiten lassen.
Gut beobachten lässt sich der positive Effekt der Nutzung eines vortrainierten Modells zu Initialisierung der Netzwerkparameter. So erreichen alle getesteten Konfigurationen bereits nach der ersten Trainingsepoche eine Validierungsgenauigkeit von über 85% (vgl. Abb.4.1a). Die initialisierten Parameter müssen nur noch geringfügig angepasst werden, um die neue Trainingsaufgabe zu lösen, weshalb die Modelle schnell große Fortschritte erzielen.



a) Validierungsgenauigkeit nach erster Trainingsepoche b) Validierungsgenauigkeit nach finaler Trainingsepoche

Abbildung 4.1: Erreichte Klassifikationsgenaugikeit nach der ersten bzw. letzten Epoche des Fine-Tunings, der getesteten Konfigurationen

Betrachtet man die Ergebnisse der Testläufe in Kombination mit den dazugehörigen Hyperparametern (siehe Abb.4.2), so scheint die Anzahl an Trainingsepochen keinen signifikanten Einfluss auf die erziel-

[4]Obwohl zum Vergleich der Konfigurationen der Validierungsfehler berechnet wurde, wird die Modellperformanz im Folgenden über die Validierungsgenauigkeit dargestellt, da diese Metrik intuitiver ist.

te Validierungsgenauigkeit zu haben. Dies deckt sich mit der Annahme, dass sich Netzwerkparameter mittles Fine-Tuning in wenigen Epochen optimieren lassen. Betrachtet man den Trainingsverlauf des Fine-Tunings (vgl. Abb. 4.4), so stellt man fest, das für die meisten Konfigurationen nach $10 - 15$ Epochen keine großen Verbesserungen mehr im Hinblick auf die Validierungsgenauigkeit erzielt werden.

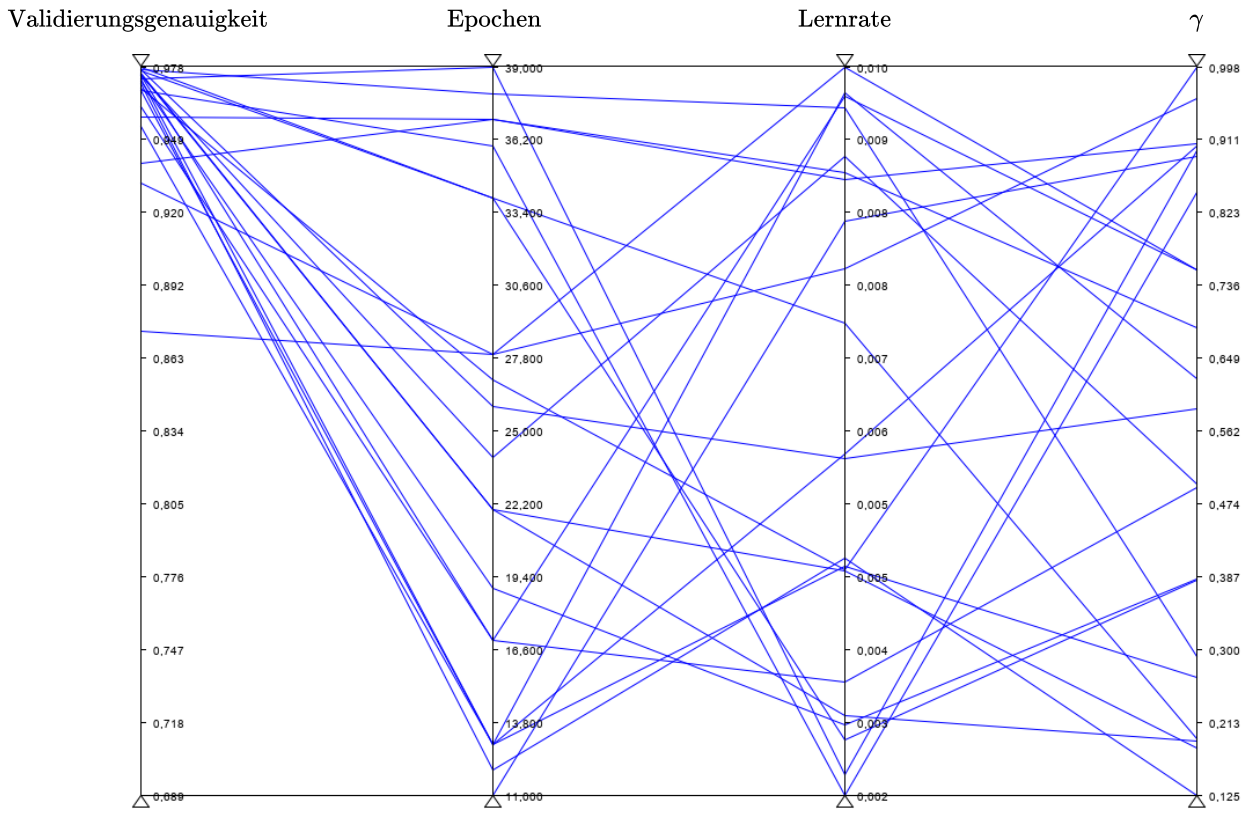


Abbildung 4.2: Parallele Darstellung der getesteten Konfigurationen des Fine-Tunings. Jede Linie repräsentiert einen Konfiguration und die von ihr erzielte Validierungsgenauigkeit.

Beleuchtet man die in den getesteten Konfigurationen genutzten Lernraten, so stellt sich heraus, dass alle Testläufe mit einer Lernrate unter 0.005 eine sehr gute Validierungsgenauigkeit erreichen. Werden höhere Lernraten genutzt, so unterscheiden sich die erzielten Ergebnisse deutlich stärker. Bezieht man die dazugehörigen γ-Faktoren mit ein, sieht man, dass Konfigurationen mit hoher Lernrate, aber niedrigem γ-Faktor, also mit starker Reduktion der Lernrate während des Trainingsverlaufes, ebenfalls sehr gute Ergebnisse erzielen. Läufe mit hoher Lernrate, sowie hohem γ-Faktor schneiden dagegen eher schlechter ab (Siehe Abb. 4.3). Analysiert man die Trainingsverläufe der unterschiedlichen Konfigurationen (Siehe Abb.4.4), findet sich eine mögliche Erklärung für diese Verhalten. Bei niedriger Lernrate nähert sich die Validierungsgenauigkeit ohne starke Einbrüche einem Maximum an. Ist die Lernrate hoch fluktuiert die Validierungsgenauigkeit jedoch stark. Dies weist darauf hin, dass die Modelle nicht in der Lage sind ein stabiles Optimum für ihre Parameter zu finden. Hohe Lernraten können dazu führen, dass Parameter bei Optimierungsschritten zu stark verändert werden und so ihr Optimum immer wieder überspringen. Nach Abschluss der ersten 10 Epochen wird die Lernrate mit dem γ-Faktor multipliziert. Für Läufe mit hoher initialer Lernrate und niedrigem γ-Faktor lässt ab diesem Punkt ein gleichmäßigerer Lernprozess beobachten.

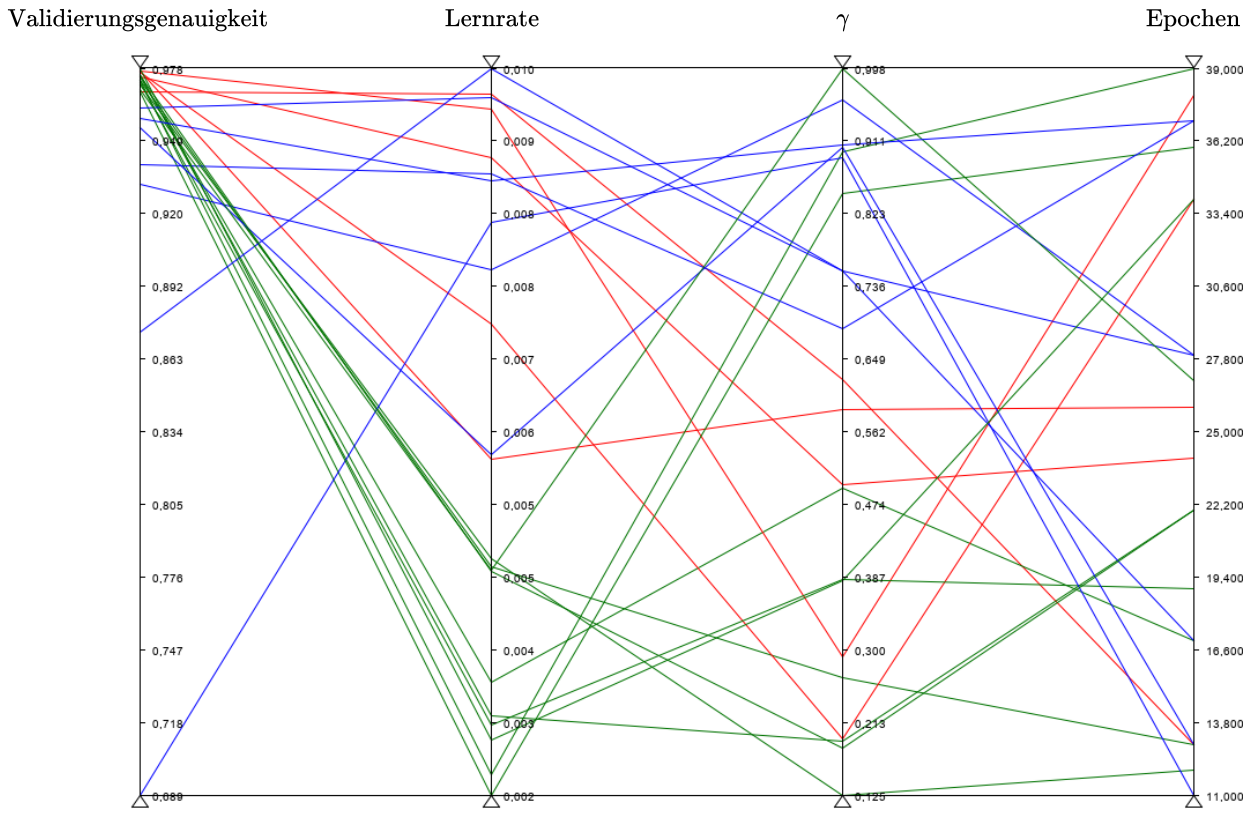


Abbildung 4.3: Betrachtung der genutzten Lernraten und γ-Faktoren. Grün: Konfigurationen mit niedriger Lernrate (< 0.005), Rot: Hohe Lernrate und niedriges γ (< 0.65), Blau: Hohe Lernrate und hohes γ

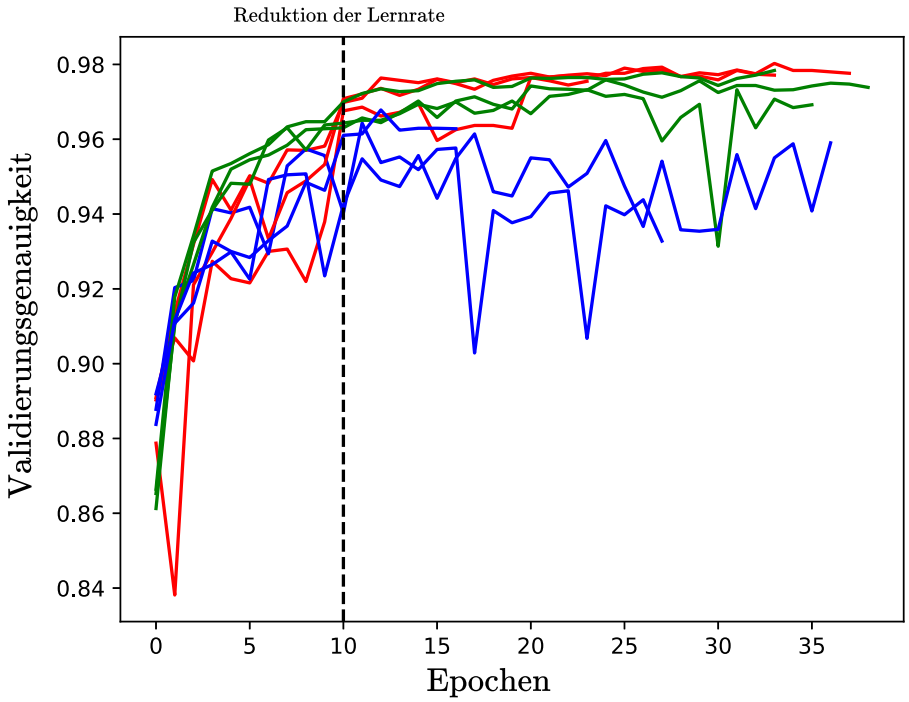
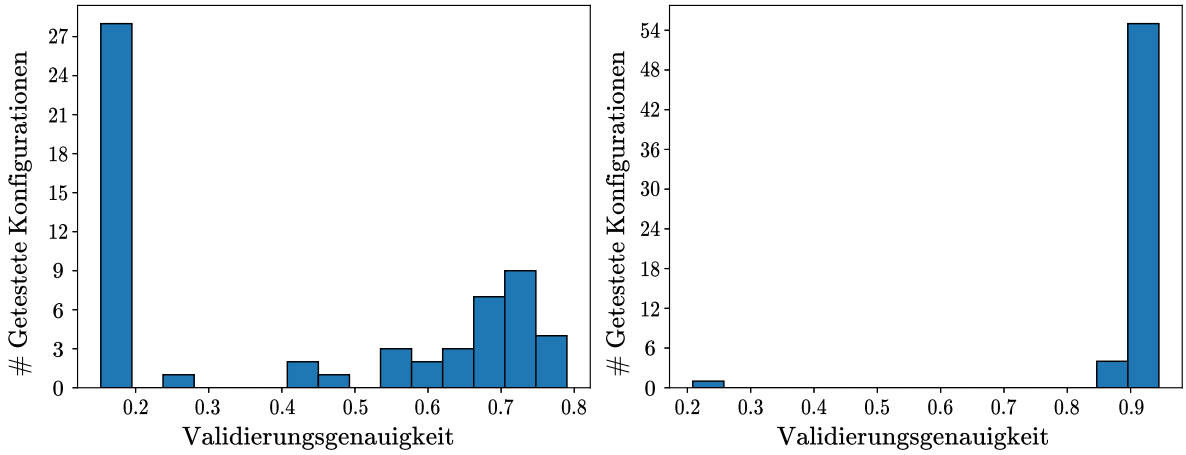


Abbildung 4.4: Trainingsverlauf der Fine-Tunings, bei unterschiedlicher Lernrate und γ. Farbgebung analog zu Abb.4.3

Die Konfiguration mit dem besten Trainingsergebnis schreibt eine Trainingsdauer von 22 Epoche, bei einer initialen Lernrate von 0.0032 und einem γ-Faktor von 0.19 vor und erzielt nach der letzten Epoche eine Validierungsgenauigkeit von 97.6%. Das dabei trainierte Modell ist der Ausgangspunkt für die Hyperparameteroptimierung des Attention-Trainings. Analog wie in den Experimenten zum Fine-Tuning werden mit Hilfe von NNOPT 60 unterschiedliche Konfigurationen für das Attention-Training getestet.



a) Validierungsgenauigkeit nach erster Trainingsepoche b) Validierungsgenauigkeit nach finaler Trainingsepoche

Abbildung 4.5: Erreichte Klassifikationsgenaugikeit nach der ersten bzw. letzten Epoche des Attention-Trainings, der getesteten Konfigurationen

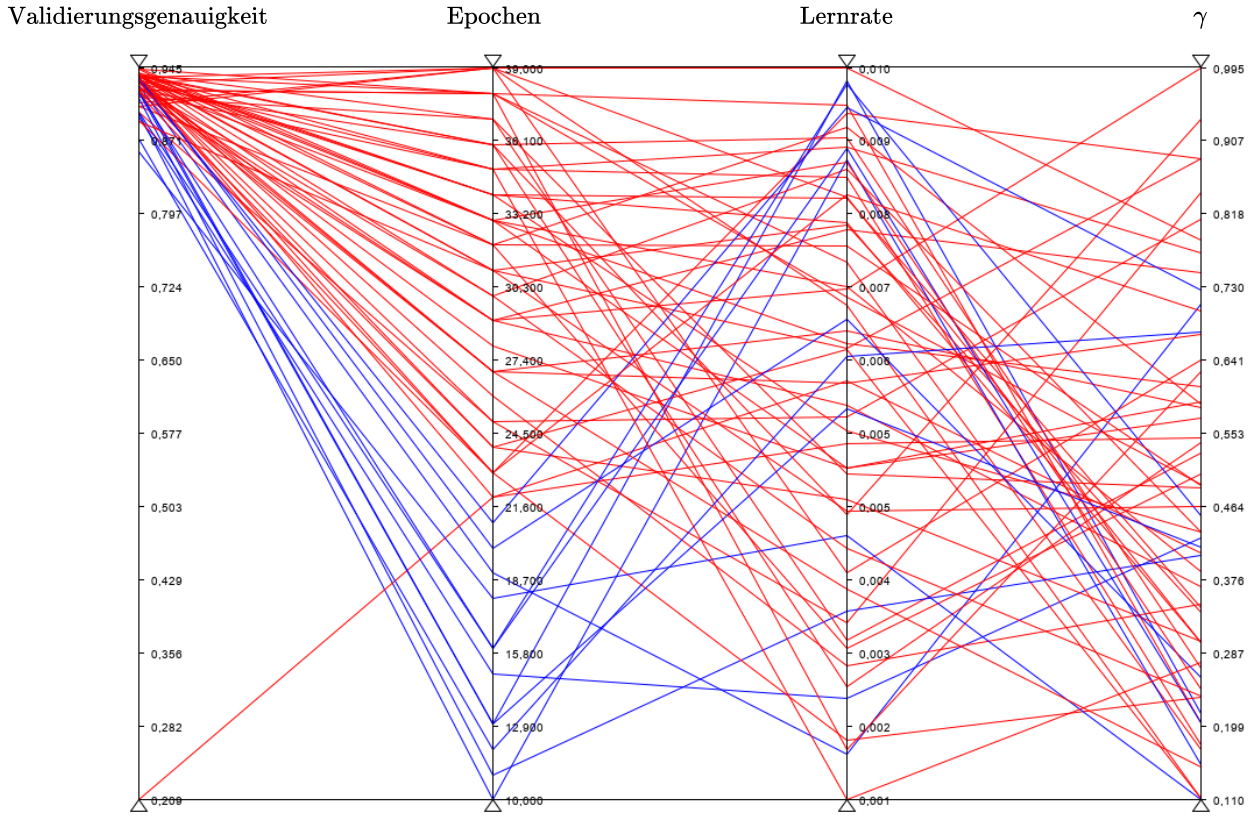


Abbildung 4.6: Betrachtung der gewählten Trainingsdauer. Rot: Über 20 Epochen, Blau: 20 oder weniger Epochen

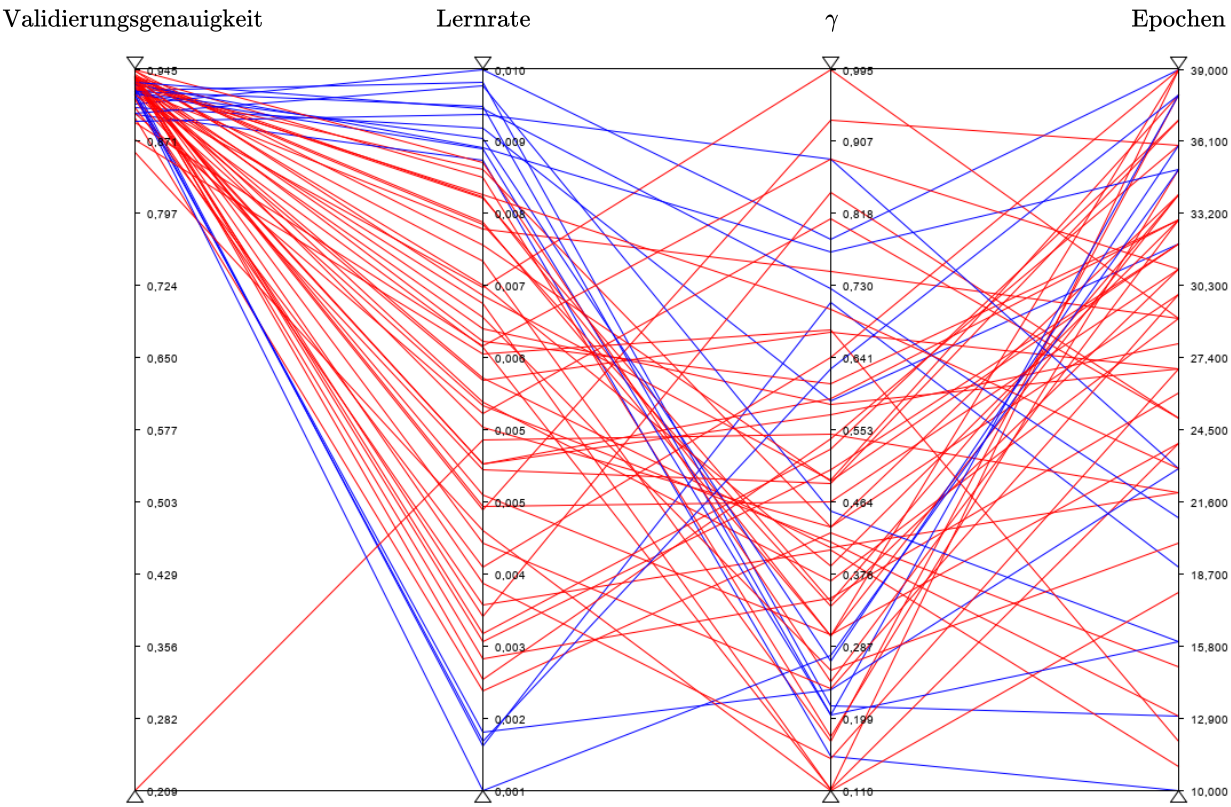


Abbildung 4.7: Betrachtung der Lernrate. Rot: Bereich zwischen 0.0017 und 0.0088

Literaturverzeichnis

[1] Ferdinand Maiwald, Jonas Bruschke, Christoph Lehmann, and Florian Niebling. A 4D Information System for the Exploration of Multitemporal Images and Maps using Photogrammetry, Web Technologies and VR/AR. *Virtual Archaeology Review*, 10:1, 07 2019.

[2] Hyeonwoo Noh, Andre Araujo, Jack Sim, Tobias Weyand, and Bohyung Han. Large-Scale Image Retrieval with Attentive Deep Local Features. pages 3476–3485, 10 2017.

[3] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[4] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Lost in Quantization: Improving Particular Object Retrieval in Large Scale Image Databases. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[5] Yan-Tao Zheng, Ming Zhao, Yang Song, Hartwig Adam, Ulrich Buddemeier, Alessandro Bissacco, Fernando Brucher, Tat-Seng Chua, Hartmut Neven, and Jay Yagnik. Tour the World: A Technical Demonstration of a Web-Scale Landmark Recognition Engine. In *Proceedings of the 17th ACM International Conference on Multimedia*, MM '09, page 961–962, New York, NY, USA, 2009. Association for Computing Machinery.

[6] Arnold W. M. Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. Content-Based Image Retrieval at the End of the Early Years. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(12):1349–1380, December 2000.

[7] David Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60:91–, 11 2004.

[8] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded up Robust Features. volume 3951, pages 404–417, 07 2006.

[9] Yan Ke and Rahul Sukthankar. PCA-SIFT: A more Distinctive Representation for Local Image Descriptors. volume 2, pages II–506, 05 2004.

[10] Cordelia Schmid and J. Ponce. Semi-Local Affine Parts for Object Recognition. *BMVC04*, 08 2004.

[11] Miaojing Shi, Yannis Avrithis, and Hervé Jégou. Early Burst Detection for Memory-Efficient Image Retrieval. 06 2015.

[12] Matthew Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Neural Networks. volume 8689, 11 2013.

[13] Artem Babenko, Anton Slesarev, Alexandr Chigorin, and Victor Lempitsky. Neural Codes for Image Retrieval. volume 8689, 04 2014.

[14] Ali S. Razavian, Josephine Sullivan, Stefan Carlsson, and Atsuto Maki. Visual Instance Retrieval with Deep Convolutional Networks. *ITE Transactions on Media Technology and Applications*, 4(3):251–258, 2016.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems*, 25, 01 2012.

[16] Lingxi Xie, Richang Hong, Bo Zhang, and Qi Tian. Image classification and retrieval are one. In *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, ICMR '15, page 3–10, New York, NY, USA, 2015. Association for Computing Machinery.

[17] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv 1409.1556*, 09 2014.

[18] J. R. Uijlings, K. E. Sande, T. Gevers, and A. W. Smeulders. Selective Search for Object Recognition. *Int. J. Comput. Vision*, 104(2):154–171, September 2013.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. 7, 12 2015.

[20] Filip Radenović, Giorgos Tolias, and Ondřej Chum. CNN Image Retrieval Learns from BoW: Unsupervised Fine-Tuning with Hard Examples. volume 9905, pages 3–20, 10 2016.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems*, 25, 01 2012.

[22] Andrew McCallum and Kamal Nigam. A comparison of event models for naive bayes text classification. In *AAAI 1998*, 1998.

[23] James B. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. 1967.

[24] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Perez. Aggregating Local Descriptors into a Compact Image Representation. pages 3304 – 3311, 07 2010.

[25] Jingdong Wang, Heng Shen, Jingkuan Song, and Jianqiu Ji. Hashing for Similarity Search: A Survey. 08 2014.

[26] Tobias Weyand, Andre Araujo, Bingyi Cao, and Jack Sim. Google Landmarks Dataset v2 – A Large-Scale Benchmark for Instance-Level Recognition and Retrieval, 2020.

[27] Shuhei Yokoo, Kohei Ozaki, Edgar Simo-Serra, and Satoshi Iizuka. Two-stage Discriminative Reranking for Large-scale Landmark Retrieval, 2020.

[28] Hervé Jégou and Ondřej Chum. Negative Evidences and Co-occurences in Image Retrieval: The Benefit of PCA and Whitening. pages 774–787, 10 2012.

[29] Jerome Friedman, Jon Bentley, and Raphael Finkel. An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Trans. Math. Softw.*, 3:209–226, 09 1977.

[30] Piotr Indyk. Nearest Neighbors in High-Dimensional Spaces. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry, Second Edition*, pages 877–892. Chapman and Hall/CRC, 2004.

[31] Martin A. Fischler and Robert C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[32] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product Quantization for Nearest Neighbor Search. *IEEE transactions on pattern analysis and machine intelligence*, 33:117–28, 01 2011.

[33] J. Bergstra, D. Yamins, and D. D. Cox. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page I–115–I–123. JMLR.org, 2013.