Inaugural-Dissertation

zur
Erlangung der Doktorwürde
der
Naturwissenschaftlich-Mathematischen Gesamtfakultät
der
Ruprecht-Karls-Universität
Heidelberg

vorgelegt von

Philipp David Lösel

aus Bruchsal

Tag der mündlichen Prüfung:

GPU-basierte Verfahren zur Segmentierung biomedizinischer Bilddaten

18. November 2021

Betreuer: Prof. Dr. Vincent Heuveline
Co-Betreuer: Prof. Dr. Peter Bastian

Zusammenfassung

Die Analyse medizinischer und biologischer Bilddaten erfordert häufig die Isolierung einzelner Strukturen aus einem 3D-Volumen durch Segmentierung. Trotz einer Vielzahl halb- und vollautomatischer Verfahren erfolgt die Bildsegmentierung oft noch manuell und gilt nach wie vor für viele Szenarien als die arbeitsintensivste und zeitaufwendigste Aufgabe innerhalb der 3D-Bildanalyse. Die herkömmliche manuelle Segmentierung vieler Schichten, gefolgt von einer linearen Interpolation und einer manuellen Korrektur der Ergebnisse verhindert jedoch in vielen Fällen die Analyse einer großen Anzahl von Proben.
In dieser Arbeit wird darum ein neues parameterfreies Verfahren zur halbautomatischen Segmentierung großer komplexer 3D-Bilddaten entwickelt, das auf einer Interpolation von wenigen manuell vorsegmentierten Schichten basiert, wobei es den gesamten zugrunde liegenden volumetrischen Bilddatensatz berücksichtigt. Die Interpolation erfolgt durch gewichtete Random Walks, deren Unabhängigkeit voneinander eine Berechnung durch massiv parallele Hardware, wie den Grafikprozessoren (GPUs), ermöglicht. Anhand einer vielfältigen Auswahl von Beispieldatensätzen wird gezeigt, dass dieses GPU-basierte Verfahren, insbesondere bei der Segmentierung sehr großer komplexer Bilddaten, wie sie typischerweise in der Mikro-Computertomographie (μCT) entstehen, dazu in der Lage ist, sowohl den Zeit- und Arbeitsaufwand erheblich zu reduzieren als auch die Qualität der Ergebnisse deutlich zu steigern. Dabei erreicht es eine deutliche Beschleunigung und höhere Genauigkeit gegenüber dem konventionellen Ansatz der fast ausschließlich manuellen Segmentierung und auch gegenüber den am weitesten verbreiteten Segmentierungsalgorithmen.
Die GPU-basierten Random Walks sind insbesondere dann geeignet, wenn wenig Vorwissen über das zu segmentierende Objekt vorhanden ist, zum Beispiel bei der Beschreibung einer neu entdeckten Art, oder zum Erstellen von Trainingsdaten für ein anschließendes maschinelles Lernen. Um darüber hinaus bei der Segmentierung vieler ähnlicher Proben eine weitgehend automatische Segmentierung zu ermöglichen, wird hier zusätzlich ein auf künstlichen neuronalen Netzen basierendes Verfahren vorgestellt und anhand von 110 μCT-Scans von Honigbienengehirnen evaluiert.
Ergänzend wird auf Basis der entwickelten Algorithmen die neue Online-Segmentierungsplattform Biomedisa präsentiert. Die Plattform ist über einen Webbrowser zugänglich und erfordert keine komplexe und langwierige Konfiguration von Software- und Modellparametern. Sie richtet sich gezielt an die Bedürfnisse von Wissenschaftlern/-innen, die nicht über umfangreiche Computer- und Softwarekenntnisse verfügen. Die integrierten GPU-basierten Methoden ermöglichen eine intuitive Anwendung für verschiedene bildgebende Verfahren innerhalb eines breiten Spektrums wissenschaftlicher Disziplinen. Die Plattform wurde bereits in mehreren Studien erfolgreich eingesetzt. Darüber hinaus ermöglicht ihr modularer Aufbau eine leichte Erweiterung der hier vorgestellten Kernfunktionen um weitere benutzerspezifische Funktionalitäten.

Abstract

The analysis of medical and biological image data often requires the isolation of individual structures from a 3D volume by means of segmentation. Despite a large number of semi-automatic and fully automatic methods, image segmentation is often still done manually and is still considered the most labor-intensive and time-consuming task within 3D image analysis for many scenarios. In particular, the analysis of a large quantity of samples is often severely limited by the traditional manual segmentation of many slices with subsequent linear interpolation and manual correction of the results.

In this thesis, a new parameter-free method for the semi-automatic segmentation of large and complex 3D image data is developed. The segmentation is based on a smart interpolation of sparsely pre-segmented slices, taking into account the entire underlying volumetric image data. The interpolation is carried out using weighted random walks. The independence of the random walks enables a calculation using massively parallel computer architectures such as graphics processing units (GPUs). Using a diverse selection of sample datasets, it is shown that this GPU-based method can drastically reduce both time and human effort required for segmenting large images, as are typically generated by micro-computed tomography (μCT). It achieves a significant acceleration and higher accuracy compared to the conventional approach of almost exclusively manual segmentation as well as to most widespread semi-automatic segmentation algorithms.

The GPU-based random walks are particularly valuable when there is little *a priori* knowledge about the object to be segmented, for example when describing a newly discovered species or for creating training data for subsequent machine learning. In order to enable a largely automatic segmentation when segmenting many similar samples, a method based on artificial neural networks is also presented here and evaluated using 110 μCT scans of honeybees' brains.

In addition, based on the algorithms developed, the new online segmentation platform Biomedisa will be presented. The platform is accessible via web browser and does not require complex and tedious software configuration or parameter optimization. It is specifically aimed at the needs of scientists who do not have extensive computer and software knowledge. The integrated GPU-based methods enable intuitive application to various imaging modalities in a wide range of scientific disciplines. The platform has already been used successfully in several studies. Moreover, due to the modular structure of the platform, its core functions can easily be expanded to include additional user-specific functions.

Danksagung

Diese Arbeit entstand im Rahmen meiner Anstellungen am Interdisziplinären Zentrum für Wissenschaftliches Rechnen (IWR) der Ruprecht-Karls-Universität Heidelberg, in den durch das Bundesministerium für Bildung und Forschung (BMBF) geförderten Projekten ASTOR (05K2013) und NOVA (05K2016), sowie am Heidelberger Institut für Theoretische Studien (HITS). Mein vorrangiger Dank gilt darum dem BMBF und der Klaus Tschira Stiftung (KTS), ohne deren finanzielle Unterstützung diese Arbeit nicht möglich gewesen wäre, sowie der Fakultät für Mathematik und Informatik der Ruprecht-Karls-Universität Heidelberg für meine Annahme als Doktorand.
Mein besonderer Dank gilt Herrn Prof. Dr. Vincent Heuveline für das entgegengebrachte Vertrauen, für seine wissenschaftliche Anleitung in jeder Phase der Entstehung dieser Dissertation und für all die Möglichkeiten, die ich bekommen habe, um meine Forschung voranzutreiben.
Ich danke Dr. Thomas van de Kamp für den regen wissenschaftlichen Austausch in den Jahren gemeinsamer Forschung und für einige in dieser Arbeit enthaltenen Grafiken, meinen Kollegen in den Arbeitsgruppen Engineering Mathematics and Computing Lab (EMCL) und Data Mining and Uncertainty Quantification (DMQ) für eine motivierende und inspirierende Arbeitsatmosphäre, Dr. Philipp Gerstner für die inhaltliche Korrektur großer Teile dieser Dissertation, Alejandra Jayme für ihre technische Expertise und Lydia Mehra für ihre fortwährende Unterstützung und ihre redaktionelle Hilfe. Außerdem war es mir eine große Freude mit Dr. Coline Monchanin zusammenzuarbeiten.
Schließlich danke ich meiner Familie und meinen Freunden für ihre weisen Ratschläge und ihre mitfühlenden Ohren, die mir anregende Diskussionen und freudige Ablenkungen boten, um meine Gedanken außerhalb meiner Arbeit auszuruhen, meinem Vater dafür, dass er mir viel von dem beigebracht hat, was ich heute über (wissenschaftliches) Schreiben weiß, Isabel für ihr Interesse und ihre Unterstützung zu Beginn meiner wissenschaftlichen Zeit und Gergana für ihr sorgfältiges und sensibles Lektorat und für ihre unschätzbare Hilfe bei der Fertigstellung dieser Dissertation.

Für Mama

Inhaltsverzeichnis

1 Einleitung

Die dreidimensionale (3D) Bildgebung treibt in vielen wissenschaftlichen Disziplinen, wie der Medizin [Aerts et al., 2014], der Grundlagenbiologie [Sigal et al., 2018, de Bakker et al., 2016], der Paläontologie [Qvarnström et al., 2021] und der Archäologie [Harvati et al., 2019] den Fortschritt erheblich voran. Zerstörungsfreie bildgebende Verfahren wie die Mikro-Computertomographie (μCT) oder die Magnetresonanztomographie (MRT) ermöglichen dabei die Darstellung feinskaliger interner Strukturen sowohl in vivo als auch ex vivo. Zur Analyse medizinischer und biologischer Bilddaten wird häufig die Isolierung einzelner Strukturen aus dem 3D-Volumen durch eine sogenannte Segmentierung benötigt. Ein Bild lässt sich dabei als Funktion

$$u\colon \Omega \to \mathcal{M}^k \text{ mit } \Omega \subset \mathbb{R}^d,$$

auffassen, wobei das Gebiet Ω häufig ein Rechteck ($d = 2$) oder ein Quader ($d = 3$) ist und die Zielmenge $\mathcal{M}$ die Farbtiefe des Bildes vorgibt. Für $k = 1$ handelt es sich um ein Graustufenbild und für $k = 3$ um ein Farbbild basierend auf dem RGB-Farbraum mit drei Kanälen. Diese Definition schließt auch bewegte Bilder mit ein, wobei hier eine Dimension der Zeit zugeordnet wird. Zum Beispiel erhält man für 3D + t ein vierdimensionales Objekt ($d = 4$). Die Bilddaten liegen im Allgemeinen in Form eines Tensors vor. Dabei wird das Gebiet Ω in viele gleichgroße Rechtecke (Pixel für $d = 2$) oder Würfel (Voxel für $d = 3$) unterteilt, auf denen das Bild jeweils einen einheitlichen Wert annimmt. Im zweidimensionalen Fall gilt für die Zerlegung:

$$\overline{\Omega} = \bigcup_{\substack{1\le i\le m\\ 1\le j\le n}} \overline{\Omega}_{ij},$$

mit $\Omega_{ij} \cap \Omega_{i'j'} = \varnothing$ falls $(i, j) \neq (i', j')$ und $u|_{\Omega_{ij}} = const.$ Zum Beispiel erhält man für die Zielmenge $\mathcal{M} = \{0, \ldots, 255\}$ und $k = 1$ ein Graustufenbild mit einer Farbtiefe von 8-Bit. Dabei wird jedem der $m \times n$ Pixel Ω_{ij} ein ganzzahliger Wert zugewiesen. Das heißt:

$$u|_{\Omega_{ij}} \in \{0, \ldots, 255\},$$

wobei der Wert 0 schwarz, der Wert 255 weiß und die Werte dazwischen als Graustufen von dunkel nach hell dargestellt werden. Häufig wird auch eine Farbtiefe von 16-Bit oder 32-Bit mit den Zielräumen $\{0, \ldots, 2^{16} - 1\}$ bzw. $\{0, \ldots, 2^{32} - 1\}$ oder eine Darstellung durch Gleitkommazahlen mit einfacher oder doppelter Genauigkeit auf dem Intervall 0 bis 1 verwendet, wobei der Speicherplatzbedarf entsprechend ansteigt. Eine Segmentierung

$$A\colon \Omega \to \{0, 1, 2, \ldots\}$$

weist nun jedem Pixel Ω_{ij} des Bildes eine Klasse (oder auch *Label*) zu. Im einfachsten Fall wird dabei das Bild in zwei Regionen, das zu segmentierende Objekt und den Hintergrund,

unterteilt. Die Segmentierung kann selbst wieder als Bild, in diesem Fall als ein Binärbild

$$A|_{\Omega_{ij}} = \begin{cases} 1, & \text{falls } \Omega_{ij} \text{ dem Objekt zugeordnet wird,} \\ 0, & \text{falls } \Omega_{ij} \text{ dem Hintergrund zugeordnet wird,} \end{cases}$$

von gleicher Größe aufgefasst werden. Die automatische Umsetzung dieser dreidimensionalen Rekonstruktion der in den Bilddaten enthaltenen Strukturen ist jedoch in vielen Fällen ein immer noch nicht zufriedenstellend gelöstes Problem. Kontinuierliche technologische Verbesserungen der bildgebenden Verfahren führen zu immer höher werdenden Auflösungen, kürzeren Aufnahmezeiten und einer damit einhergehenden, rasant ansteigenden Anzahl an zugleich immer größer werdenden Bilddaten [Maire and Withers, 2014, dos Santos Rolo et al., 2014]. Exemplarisch ist hier die Zunahme der durch das Aufkommen der Synchrotron-Röntgen-Mikrotomographie (SR-μCT) erzeugten Datenmengen. Der stetige technologische Fortschritt erhöht zugleich die Nachfrage nach einer beschleunigten Bildanalyse. Insbesondere die Bildsegmentierung ist dabei jedoch oft der arbeitsintensivste und fehleranfälligste Arbeitsschritt innerhalb der 3D-Bildanalyse.
Ein vielversprechender Ansatz für eine beschleunigte Bildsegmentierung basiert auf den neuesten Entwicklungen im Bereich künstlicher neuronaler Netzwerke [Litjens et al., 2017]. Jedoch hängt der Erfolg dieser Verfahren maßgeblich von der Anzahl der zur Verfügung stehenden und in der Regel manuell erstellten Trainingsdaten ab. Darüber hinaus können künstliche neuronale Netze, wie auch andere Verfahren des maschinellen Lernens, nur auf sich wiederholenden Strukturen wie Organe [Christ et al., 2016], Tumore [Akkus et al., 2017], Zellen [Stegmaier et al., 2016] oder Modellorganismen [Panser et al., 2016, Weinhardt et al., 2018] angewandt werden und sind nach erfolgtem Training nicht ohne Weiteres auf andere Strukturen oder Objekte übertragbar.
Sind jedoch nur wenige Informationen über das zu segmentierende Objekt *a priori* vorhanden, sind automatisierte Routinen kaum anwendbar. In diesen Fällen gilt die manuelle Segmentierung eines/einer, auf dem jeweiligen Forschungsbereich spezialisierten Experten/-in, gefolgt von einer linearen Interpolation und einer anschließenden manuellen Korrektur der Ergebnisse als die übliche und erfolgversprechendste Vorgehensweise [Dumbravă et al., 2016, Pardo et al., 2017, Gross et al., 2019, Jones et al., 2019]. Dabei werden in einzelnen Schichten des dreidimensionalen Volumens den verschiedenen Strukturen manuell Label zugewiesen. Je nach Verlauf und Lage der Strukturen müssen die Abstände zwischen den Schichten größer oder kleiner gewählt werden. Anschließend erfolgt eine Interpolation der Label, um das gesuchte dreidimensionale Objekt zu erhalten. In der Regel erfolgt die Interpolation dabei ausschließlich auf Basis der Label. Die zugrunde liegenden Bilddaten bleiben bei der Interpolation häufig unberücksichtigt, was einen erheblichen Informationsverlust zur Folge hat. Diverse frei verfügbare und kommerzielle Softwarelösungen können für solch eine morphologische Interpolation genutzt werden, bspw. MITK [Wolf et al., 2005], ITK-SNAP [Yushkevich et al., 2006], ImageJ/Fiji [Schindelin et al., 2012], 3D Slicer [Kikinis et al., 2014], Microscopy Image Browser [Belevich et al., 2016], Amira/Avizo (Thermo Fisher Scientific, Waltham, USA), MeVisLab (MeVis Medical Solutions AG, Bremen, Germany) und Dragonfly (Object Research Systems, Montreal, Kanada).
Eine auf diese Weise erstellte Segmentierung benötigt für ein akkurates Ergebnis eine Vorsegmentierung sehr eng beieinander liegender Schichten. In manchen Fällen muss sogar jede

Schicht manuell segmentiert werden. Dieser konventionelle Segmentierungsansatz ist daher äußerst zeitaufwendig, umständlich und erschwert die Auswertung sehr großer Bilddaten oder sehr vieler stark unterschiedlicher Proben erheblich. Darüber hinaus beeinflussen Artefakte wie Linienartefakte oder stark geglättete Segmentierungsergebnisse, wie sie typischerweise bei einer morphologischen Interpolation manuell segmentierter Schichten mit anschließender Nachbearbeitung entstehen, die Qualität und Attraktivität der Ergebnisse. Feine Strukturen, wie Haare, lassen sich auf diese Weise kaum oder nur in ungenügender Weise rekonstruieren.
Neben dieser vorwiegend manuellen Vorgehensweise wurde bereits eine Vielzahl weiterer semi-automatischer Verfahren für unterschiedlichste Anwendungen eingesetzt. Eine Interaktion wird hier beispielsweise durch das initiale Setzen einer Kontur [Chan and Vese, 2001, Marquez-Neila et al., 2014] oder einer Bounding Box [Rother et al., 2004] erreicht. Die Initialisierung der Segmentierung durch sogenannte Saatpunkte (d. h. durch von den Benutzern/-innen vereinzelt manuell zugewiesene Pixel) ist aufgrund der intuitiven Handhabung ebenfalls äußerst beliebt. Insbesondere sind hier die Ansätze Graph Cut (GC) [Boykov and Jolly, 2001], GrowCut, Geodätische Distanzen (GeoS) [Bai and Sapiro, 2007, Criminisi et al., 2008] und Random Walker Algorithmus (RW) [Grady, 2006] zu nennen. Der Einsatz von Methoden des maschinellen Lernens wie in *WEKA* [Arganda-Carreras et al., 2017], *ilastik* [Berg et al., 2019] und *Slic-Seg* [Wang et al., 2016] beschränkt sich meist auf charakteristische Strukturen wie Zellen und Fasern und erfordert das Training benutzerdefinierter Merkmale, deren erfolgreiche Auswahl stark von der nutzerspezifischen Erfahrung abhängt.
Anstelle einer rein morphologischen Interpolation lassen sich diese interaktiven bzw. semi-automatischen Verfahren ebenfalls zur Interpolation manuell vorsegmentierter Schichten verwenden. Dabei werden die Pixel der vorsegmentierten Schichten als Saatpunkte betrachtet. Die korrekte Konfiguration und Implementation dieser Methoden stellt jedoch viele Wissenschaftler/-innen ohne tiefere Kenntnisse der Verfahren und im Umgang mit Software vor große Herausforderungen. Die Implementierungen von GeoS in *GeodisTK*, GC in *MedPy*, und RW in der *scikit-image* Softwarebibliothek [van der Walt et al., 2014] sind zwar weit verbreitet, benötigen jedoch Programmierkenntnisse. Darüber hinaus sind die Ergebnisse insbesondere der Verfahren GC und RW abhängig von einem Modellparameter, dessen ideale Wahl stark von Anwendung zu Anwendung variiert und häufig durch *Trial and Error* bestimmt werden muss. Beide Verfahren haben zusätzlich einen sehr hohen Speicherbedarf, was ihre Anwendung auf sehr große Datensätze zusätzlich erschwert. Aktive Konturen hängen in der Regel gleich von mehreren frei wählbaren Parametern ab. Außerdem kann hier immer nur ein Objekt segmentiert werden, was den Aufwand bei einer großen Anzahl von Label deutlich erhöht. Diese Hindernisse erschweren den Zugang vieler nicht IT-affiner Wissenschaftler/-innen zu diesen Verfahren. Aus diesem Grund greifen Biologen/-innen und Mediziner/-innen häufig auf die traditionelle Methode einer rein manuellen Segmentierung von Schichten mit anschließender morphologischer Interpolation zurück.
Das explizite Ziel dieser Dissertation ist es, ein auf viele unterschiedliche Szenarien anwendbares, nutzerfreundliches Verfahren zu entwickeln, das die manuelle Segmentierung von Objekten unbekannter Morphologie, enthalten in sehr großen volumetrischen Bildda-

ten, signifikant erleichtert, beschleunigt und darüber hinaus das Segmentierungsergebnis gegenüber der traditionellen Vorgehensweise und auch gegenüber vergleichbarer Verfahren qualitativ wesentlich verbessert.
Das im Rahmen dieser Arbeit neu entwickelte Verfahren nutzt gewichtete Random Walks und wurde speziell für eine massiv parallele Berechnung durch Grafikprozessoren entworfen (GPUs, *Graphics Processing Units*), um den stetig größer werdenden Volumen Rechnung zu tragen. Dabei kann das Verfahren, wie die vorher genannten semi-automatischen Methoden, als Interpolation zwischen den vorsegmentierten Schichten, unter Berücksichtigung der Bilddaten, betrachtet werden. Im Gegensatz zu den zuvor genannten Verfahren bedarf es jedoch keiner Parameteroptimierung. Darüber hinaus kann die Berechnung der Random Walks durch deren gegenseitige Unabhängigkeit sehr stark parallelisiert werden. Das Verfahren eignet sich darum besonders gut für eine Berechnung durch Grafikprozessoren, wohingegen die Geschwindigkeit der anderen, seriell arbeitenden Algorithmen maßgeblich von der Frequenz der Prozessoren, in der Regel einer zentralen Recheneinheit (CPU), abhängt. Die Frequenz der CPUs wird jedoch zunehmend durch physikalische Eigenschaften begrenzt. Algorithmen müssen darum vermehrt eine parallele Struktur aufweisen, um in der Lage zu sein, die immer größer werdenden Datenmengen in einer angemessenen Zeit verarbeiten zu können. Die Methode der GPU-basierten Random Walks stützt sich somit auf einen anhaltenden Trend zur Nutzung paralleler Hardwarebeschleuniger und konnte in den vergangenen Jahren allein durch den technologischen Fortschritt eine massive Beschleunigung erfahren.
Das meist ebenfalls durch Grafikprozessoren realisierte *Deep Learning* nimmt auch in der Bildsegmentierung eine immer wichtiger werdende Rolle ein. *Deep Learning* bezeichnet hierbei das Training tiefer künstlicher neuronaler Netze zur automatischen Segmentierung einer großen Anzahl ähnlicher Proben. Auf diese Weise können beispielsweise geringfügige, aber statistisch und biologisch relevante Variationen aufgedeckt werden, die in der Lage sind, wichtige Fragen in Bezug auf Verhalten, Ökologie und Evolution der segmentierten Objekte zu beantworten, was durch die Analyse einzelner Proben nicht möglich gewesen wäre. Eine der größten Herausforderungen liegt hierbei in der Verfügbarkeit und Erzeugung großer Mengen bereits segmentierter Trainingsdaten. Die hier vorgestellten GPU-basierten Random Walks eignen sich aufgrund ihrer einfachen Nutzbarkeit, schnellen Segmentierung und hohen Genauigkeit besonders gut zur Generierung hinreichend großer Mengen Trainingsdaten. Um darüber hinaus eine automatische Segmentierung weiterer Proben ähnlicher Struktur zu ermöglichen, wird ergänzend ein Verfahren basierend auf einem künstlichen neuronalen Netz entwickelt und anhand von 110 μCT-Scans von Honigbienengehirnen evaluiert [Lösel et al., Vorb].
Darüber hinaus wurden beide Verfahren in die ebenfalls entwickelte Online-Anwendung Biomedisa (**Biomed**ical **I**mage **S**egmentation **A**pp, `https://biomedisa.org`) integriert. Die als *One-Button*-Lösung konzipierte Online-Plattform ist speziell an Wissenschaftler/-innen ohne substantielle computertechnische Kenntnisse gerichtet. Sie ist intuitiv handhabbar und über einen Webbrowser nutzbar. Als Cloud-Anwendung benötigt sie weder eine vorherige Softwareinstallation, noch die für die Verfahren notwendigen Hardwareressourcen. Für Nutzer/-innen mit kritischen, wie beispielsweise personenbezogenen Daten, steht eine *Standalone*-Version zur Verfügung. Die Anwendung wurde um einige zu-

sätzliche Funktionen erweitert. Insbesondere wurde eine strukturerhaltende Glättung der GPU-basierten Random Walks und eine Lokalisierung der Unsicherheit, mit der die Segmentierung erstellt wurde, entwickelt und in die Plattform integriert. Zudem gibt es die Möglichkeit, Ausreißer im Ergebnis zu entfernen und Löcher zu füllen. Des Weiteren lassen sich die Bilddaten und die Ergebnisse in einem *Slice Viewer* schichtweise betrachten und durch eine Rendering Software dreidimensional darstellen.
Die GPU-basierten Random Walks zur Segmentierung volumetrischer Bilddaten wurden erstmals in einer Konferenzpublikation beschrieben [Lösel and Heuveline, 2016]. Die darauf basierende Anwendung Biomedisa wurde anschließend in der multidisziplinären wissenschaftlichen Fachzeitschrift *Nature Communications* vorgestellt und sowohl als Online-Anwendung als auch als Open-Source-Software der wissenschaftlichen Gemeinschaft frei zur Verfügung gestellt [Lösel et al., 2020]. Viele Passagen dieser Dissertation beziehen sich auf diese Arbeiten. Erste Entwicklungsversionen von Biomedisa wurden bereits vor der Veröffentlichung der Plattform in einigen Studien erfolgreich eingesetzt [Mikó et al., 2018b, Lösel and Heuveline, 2017, Balanta-Melo et al., 2019, Mikó et al., 2018a, Balanta-Melo et al., 2018]. Das Verfahren spielte vor allem bei der Beschreibung parasitärer Schlupfwespen, die in mineralisierten Fliegenpuppen entdeckt wurden, eine entscheidende Rolle [van de Kamp et al., 2018]. Die detaillierte Segmentierung der Wespen war der Schlüssel zu den in der Studie durchgeführten Artenbeschreibungen, die mit einem herkömmlichen manuellen Segmentierungsansatz praktisch nicht realisierbar gewesen wären. Nach der Veröffentlichung von Biomedisa kamen, bis zum Zeitpunkt der Einreichung dieser Dissertation, zahlreiche weitere Studien hinzu [Richter et al., 2020, Gutiérrez et al., 2020, Bemmann et al., 2021, Buvinic et al., 2021, Engelkes, 2020, Richter et al., 2021a, Richter et al., 2021b, Kallinowski et al., 2021, Csader et al., 2021, Moser et al., 2021, Boudinot et al., 2021, Göttler et al., 2021].
Die hier vorliegende Arbeit gliedert sich neben der Einleitung in fünf weitere Kapitel. Kapitel 2 gibt einen Überblick über die Grundlagen der wichtigsten semi-automatischen Verfahren, die eine weite Anwendung aufweisen und häufig als Referenz verwendet werden. In Kapitel 3 wird das neu entwickelte und auf Random Walks basierende Verfahren vorgestellt und anhand verschiedener Beispieldatensätze evaluiert. Dabei wird das Verfahren der konventionellen Methode und den zuvor beschriebenen semi-automatischen Ansätzen gegenübergestellt und mit ihnen verglichen. In Kapitel 4 werden die analytischen Grundlagen künstlicher neuronaler Netze zur automatischen Bildverarbeitung aufgearbeitet und ein tiefes künstliches neuronales Netz für eine weitgehend automatisierte Segmentierung einer Vielzahl von Honigbienengehirnen eingesetzt. Darauf aufbauend wird in Kapitel 5 die Online-Anwendung Biomedisa vorgestellt. Abschließend werden die Ergebnisse in Kapitel 6 zusammengefasst und diskutiert.

2 Semi-automatische Bildsegmentierung

Dieses Kapitel gibt einen Überblick über die gängigsten halbautomatischen Segmentierungsmethoden. Dazu gehören die aktiven Konturen (Abschnitt 2.1), der Random Walker Algorithmus (Abschnitt 2.2) sowie die Methoden, die zur Bestimmung der Segmentierung den minimalen Schnitt eines Graphen (Abschnitt 2.3) oder geodätische Distanzen berechnen (Abschnitt 2.4). Alle Verfahren basieren auf einem interaktiven Prozess mit dem/der Benutzer/-in. Sie erfordern eine Initialisierung über einzelne Pixel oder ganze Regionen, eine manuelle Festlegung von Modellparametern und in der Regel eine Nachbearbeitung der Segmentierungsergebnisse. Im Abschnitt 3.7 werden die Verfahren zum Vergleich mit dem in dieser Arbeit entwickelten Verfahren zur Segmentierung großer, komplexer Datensätze verwendet.

2.1 Aktive Konturen

Aktive Konturen haben eine lange Tradition in der digitalen Bildverarbeitung [Chenyang Xu and Prince, 1998, Chan and Vese, 2001, Marquez-Neila et al., 2014, Vese and Chan, 2002]. Dabei wird eine parametrisiert oder implizit dargestellte Kurve, mittels von innen und außen wirkenden Kräften, solange bewegt, bis sich ein Gleichgewicht zwischen den Kräften einstellt und die Kontur dadurch an der gewünschten Stelle zum Stehen kommt. Die Segmentierung erfolgt durch Aufteilen des Bildes in die durch die Kontur voneinander abgegrenzten Bereiche. Die hierfür benötigten induzierten Kräfte werden aus den Bilddaten abgeleitet. Die Position, an der sich die Lösung stabilisiert, hängt in der Regel sehr stark von den gewählten Parametern ab. Sind bei diesem Ansatz, die nach außen treibenden Kräfte zu stark gewählt, entwickelt sich die Kontur über die gewünschte Stelle hinaus. Sind die Kräfte hingegen zu klein gewählt, reichen diese unter Umständen nicht aus, um lokale Minima oder stationäre Punkte zu überwinden und das Verfahren konvergiert, bevor die Kontur die gewünschte Position erreicht hat.
Traditionelle aktive Konturen basieren in der Regel auf dem Gradienten des Bildes. Hierbei erfolgt die Segmentierung an Stellen mit einem hohen Gradienten, also dort, wo ein starker Übergang von einem Bereich zu einem anderen besteht. Zum Beispiel wird beim Verfahren des *Gradient Vector Flow* [Chenyang Xu and Prince, 1998] für ein Bild $\mathcal{I}\colon \Omega \to \mathbb{R}$ durch Minimieren des Funktionals

$$F(u,v) = \int_\Omega \mu(u_x^2 + u_y^2 + v_x^2 + v_y^2) + |\nabla f|^2 |\mathbf{v} - \nabla f|^2 \, dx\, dy, \tag{2.1}$$

mit $f = |\nabla \mathcal{I}(x,y)|^2$ und $\mathbf{v}(x,y) = [u(x,y), v(x,y)]$, zunächst ein auf dem Gradienten des Bildes basierendes Vektorfeld $\mathbf{v}(x,y)$ bestimmt. Dabei zeigt der Gradient von f in Richtung der Kanten im Bild und sein Betrag ist in unmittelbarer Nähe der Kanten sehr groß und

in homogenen Bereichen des Bildes sehr klein. Folglich ist das gesuchte Vektorfeld glatt, wo $|\nabla f|$ sehr klein ist und nähert sich ∇f, wo dessen Betrag sehr groß ist. Die minimale Lösung des Funktionals 2.1 lässt sich durch Lösen der partiellen Differentialgleichungen

$$\mu\Delta u - (u - f_x)(f_x^2 + f_y^2) = 0,$$
$$\mu\Delta v - (v - f_y)(f_x^2 + f_y^2) = 0,$$

bestimmen. Anschließend kann das Vektorfeld $\mathbf{v}$ dazu genutzt werden, um eine nach der Bogenlänge parametrisierte Kurve $\mathbf{x}(s) = [x(s), y(s)]$ mit $s \in [0, 1]$ entlang des Feldes an die gewünschte Stelle zu entwickeln. Hierfür wird die Kurve $\mathbf{x}$ als Funktion von s und einer künstlich eingeführten Zeit t betrachtet, das heißt $\mathbf{x}(s, t)$. Somit lässt sich die Entwicklung der Kurve entlang $\mathbf{v}$ und unter gewissen Anforderungen an ihre Länge und Glattheit durch

$$\mathbf{x}_t(s,t) = \alpha\frac{\partial^2\mathbf{x}(s,t)}{\partial s^2} + \beta\frac{\partial^4\mathbf{x}(s,t)}{\partial s^4} + \mathbf{v}$$

beschreiben. Wenn sich die Lösung $\mathbf{x}(s, t)$ stabilisiert, das heißt, wenn $\mathbf{x}_t(s, t)$ verschwindet, dann ist die gesuchte Stelle erreicht. Die speziellen Eigenschaften von $\mathbf{v}$ ermöglichen das Auffinden von Kanten selbst dann, wenn die Kontur sehr weit vom Objekt entfernt initialisiert wird. Der Regularisierungsparameter μ muss entsprechend dem Bildrauschen gewählt werden.

Neben solch gradientenbasierten Verfahren wurden regionenbasierte aktive Konturen Modelle entwickelt, um auch dann Objekte erkennen zu können, wenn diese sich nicht durch klare Kanten hervorheben, sondern eine eher diffuse Erscheinung aufweisen, wie beispielsweise Nebel oder bei stark geglätteten Bilddaten. Bei dem im Folgenden vorgestellten Modell *Active Contours Without Edges* [Chan and Vese, 2001] wird eine auf einer Level-Set-Funktion basierende Kontur entwickelt, deren Kräfte sich aus den Bilddaten der inneren und äußeren Regionen der Kontur bestimmen lassen. Ein Vorteil dieses Ansatzes ist es, dass die Topologie der Kontur nicht an die Initialisierung gebunden ist und sich automatisch an die des Objektes anpassen kann (Abb. 2). Beispielsweise kann sich ein initial gesetzter Kreis in zwei Teile aufteilen.

Sei Ω eine beschränkte offene Teilmenge des $\mathbb{R}^2$ mit Rand $\partial\Omega$. Sei weiter $u_0 \colon \overline{\Omega} \to \mathbb{R}$ ein gegebenes Bild und $C = \{(x, y) \mid \phi(x, y) = 0\}$ die durch eine Lipschitz-stetige Funktion ϕ implizit gegebene Kontur (Abb. 1). Wie zuvor lässt sich die Entwicklung der Kontur C durch Einführen einer künstlichen Zeit t betrachten und zum Beispiel in Richtung ihrer Normalen mit Geschwindigkeit $F \in \mathbb{R}$ durch Lösen der Differentialgleichung

$$\frac{\partial\phi(t,x,y)}{\partial t} = |\nabla\phi(t,x,y)|F, \quad t \in (0,\infty), (x,y) \in \mathbb{R}^2,$$
$$\phi(0,x,y) = \phi_0(x,y), \qquad (x,y) \in \mathbb{R}^2,$$

beschreiben, wobei $\{(x, y) \mid \phi_0(x, y) = 0\}$ die initial gesetzte Kontur angibt. Ein Spezialfall ergibt sich für die durch die mittlere Krümmung induzierte Kraft $F = \operatorname{div}(\nabla\phi/|\nabla\phi|)$ (Bemerkung 3.23):

$$\frac{\partial\phi}{\partial t} = |\nabla\phi|\operatorname{div}\left(\frac{\nabla\phi}{|\nabla\phi|}\right), \quad t \in (0,\infty), (x,y) \in \mathbb{R}^2,$$
$$\phi(0,x,y) = \phi_0(x,y), \qquad (x,y) \in \mathbb{R}^2.$$

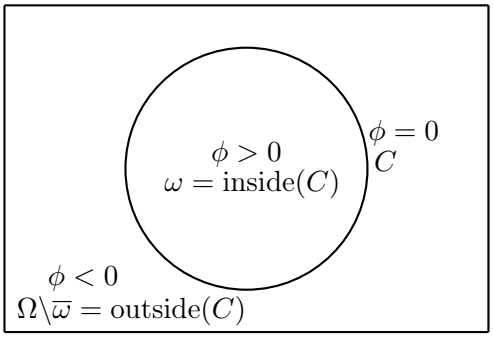

Abbildung 1: Implizite Darstellung einer Kontur mithilfe der Level-Set-Methode. Die Kontur $C = \{(x, y) \mid \phi(x, y) = 0\}$ ist gegeben durch die Nullstellenmenge der Funktion ϕ.

Sei nun die aktive Kontur C der Rand einer offenen Teilmenge $\omega \subset \Omega$, d. h. $C = \partial\omega$. Im Folgenden beschreibt $\omega = inside(C)$ den inneren Bereich und $\Omega \backslash \overline{\omega} = outside(C)$ den äußeren Bereich der Kontur C (Abb. 1), d. h.

$$\begin{aligned} C &= \{(x, y) \in \Omega \mid \phi(x, y) = 0\}, \\ inside(C) &= \{(x, y) \in \Omega \mid \phi(x, y) > 0\}, \\ outside(C) &= \{(x, y) \in \Omega \mid \phi(x, y) < 0\}. \end{aligned}$$

Zum Finden der gesuchten Kontur kann das Funktional

$$\begin{aligned} F(c_1, c_2, C) =& \mu \cdot \text{Length}(C) + \nu \cdot \text{Area}(inside(C)) \\ &+ \lambda_1 \int_{inside(C)} |u_0(x, y) - c_1|^2 \, dx \, dy \\ &+ \lambda_2 \int_{outside(C)} |u_0(x, y) - c_2|^2 \, dx \, dy, \end{aligned}$$

minimiert werden, wobei $\mu \geq 0, \nu \geq 0, \lambda_1, \lambda_2 > 0$ beliebig aber fest gewählte Parameter sind und $c_1 = average(inside)$, $c_2 = average(ouside)$. Die Regulierungsterme *Length* und *Area* wirken einer sich zu stark ausbreitenden Kontur entgegen. Häufig wird jedoch $\nu = 0$ gesetzt und die Regulierung erfolgt ausschließlich über die Länge der Kontur. Die beiden Integralterme sorgen dafür, dass durch die Entwicklung der Kontur der eingeschlossene Bereich möglichst nahe am durchschnittlichen inneren Wert c_1 und der äußere Bereich möglichst nahe am durchschnittlichen äußeren Wert c_2 liegt (Abb. 2). Durch Verwenden der Treppenfunktion

$$H(s) = \begin{cases} 1, & \text{falls } s \geq 0, \\ 0, & \text{falls } s < 0, \end{cases}$$

und der Dirac-Deltafunktion

$$\delta_0(s) = \frac{d}{ds} H(s),$$

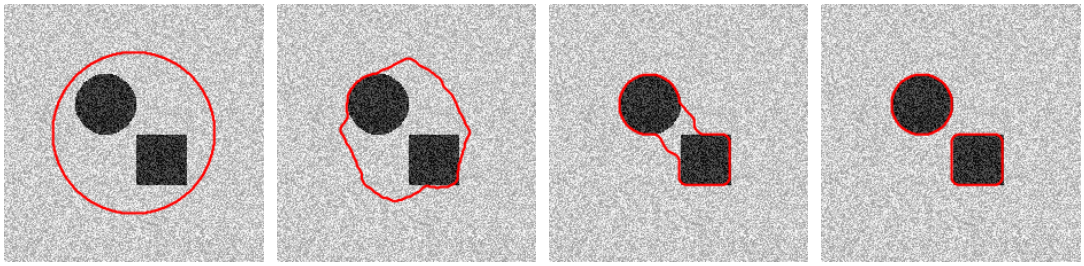

Abbildung 2: Segmentierung zweier Objekte mit einer auf der Level-Set-Methode basierenden aktiven Kontur. Die Segmentierung ist erfolgreich, wenn alle Pixelwerte im Inneren mit dem dortigen Durchschnitt c_1 und alle Pixel im Äußeren mit dem dortigen Durchschnitt c_2 übereinstimmen.

lassen sich die Regulierungsterme *Length* und *Area* über dem gesamten Gebiet Ω wie folgt beschreiben:

$$\begin{aligned}\text{Length}(C) = \text{Length}\{\phi = 0\} &= \int_\Omega |\nabla H(\phi(x,y))|\,dx\,dy,\\ &= \int_\Omega \delta_0(\phi(x,y))|\nabla\phi(x,y)|\,dx\,dy,\\ \text{Area}\{\phi \geq 0\} &= \int_\Omega H(\phi(x,y))\,dx\,dy.\end{aligned}$$

Somit lautet das Funktional

$$\begin{aligned}F(c_1,c_2,\phi) =& \mu\int_\Omega \delta_0(\phi(x,y))|\nabla\phi(x,y)|\,dx\,dy\\ &+\nu\int_\Omega H(\phi(x,y))\,dx\,dy\\ &+\lambda_1\int_\Omega |u_0(x,y)-c_1|^2 H(\phi(x,y))\,dx\,dy\\ &+\lambda_2\int_\Omega |u_0(x,y)-c_2|^2(1-H(\phi(x,y)))\,dx\,dy.\end{aligned} \tag{2.2}$$

Für festes ϕ minimieren

$$\begin{aligned}c_1 &= \frac{\int_\Omega u_0(x,y)H(\phi(x,y))\,dx\,dy}{\int_\Omega H(\phi(x,y))\,dx\,dy},\\ c_2 &= \frac{\int_\Omega u_0(x,y)(1-H(\phi(x,y)))\,dx\,dy}{\int_\Omega (1-H(\phi(x,y)))\,dx\,dy},\end{aligned}$$

das Funktional F, was den durchschnittlichen Grauwerten des Bildes innerhalb und außerhalb der Kontur entspricht. Seien im Folgenden nun c_1 und c_2 fest gewählt. Des Weiteren wird anstelle der Dirac-Deltafunktion $\delta_0(s)$ die regulierte Funktion

$$\delta_\varepsilon(s) = \frac{d}{ds}H_\varepsilon(s),$$

mit

$$H_\varepsilon(s) = \begin{cases} 1, & \text{falls } s > \varepsilon, \\ 0, & \text{falls } s < -\varepsilon, \\ \frac{1}{2}\left[1 + \frac{s}{\varepsilon} + \frac{1}{\pi}\sin\left(\frac{\pi s}{\varepsilon}\right)\right], & \text{falls } |s| \leq \varepsilon. \end{cases}$$

für $\varepsilon \to 0$ betrachtet. Für das Funktional

$$F_2[\phi] = \int_\Omega f(\phi, \nabla\phi)\, dx\, dy,$$

mit $f(s,z) = \delta_\varepsilon(s)|z|$ sei eine Funktion ϕ gesucht, die das Funktional F_2 minimiert. Für ψ beliebig und $\xi \in \mathbb{R}$ sei

$$S[\xi] \coloneqq F_2[\phi + \xi\psi] = \int_\Omega f(\phi + \xi\psi, \nabla\phi + \xi\nabla\psi)\, dx\, dy.$$

Damit F_2 für ϕ ein Extremum besitzt, muss die notwendige Bedingung

$$\frac{d}{d\xi} S[\xi]\Big|_{\xi=0} = \int_\Omega \frac{\partial f(\phi,\nabla\phi)}{\partial s}\psi + \frac{\partial f(\phi,\nabla\phi)}{\partial z}\nabla\psi\, dx\, dy = 0$$

erfüllt sein. Da $\partial f/\partial s = |z| d\delta_\varepsilon(s)/ds$ und $\partial f/\partial z = \delta_\varepsilon(s) z/|z|$ gilt:

$$\int_\Omega \frac{\partial f(\phi,\nabla\phi)}{\partial s}\psi + \frac{\partial f(\phi,\nabla\phi)}{\partial z}\nabla\psi\, dx\, dy = \int_\Omega \delta'_\varepsilon(\phi)|\nabla\phi|\psi + \delta_\varepsilon(\phi)\frac{\nabla\phi}{|\nabla\phi|}\nabla\psi\, dx\, dy = 0.$$

Mithilfe partieller Integration und der Produktregel für die Divergenz folgt:

$$\begin{aligned}
0 &= \int_\Omega \delta'_\varepsilon(\phi)|\nabla\phi|\psi + \delta_\varepsilon(\phi)\frac{\nabla\phi}{|\nabla\phi|}\nabla\psi\, dx\, dy \\
&= \int_\Omega \delta'_\varepsilon(\phi)|\nabla\phi|\psi\, dx\, dy - \int_\Omega \nabla\cdot\left(\delta_\varepsilon(\phi)\frac{\nabla\phi}{|\nabla\phi|}\right)\psi\, dx\, dy + \int_{\partial\Omega}\delta_\varepsilon(\phi)\frac{\nabla\phi}{|\nabla\phi|}\vec{n}\psi\, dx\, dy \\
&= \int_\Omega \delta'_\varepsilon(\phi)|\nabla\phi|\psi\, dx\, dy - \int_\Omega \left[\nabla\left(\delta_\varepsilon(\phi)\right)\frac{\nabla\phi}{|\nabla\phi|} + \delta_\varepsilon(\phi)\nabla\cdot\left(\frac{\nabla\phi}{|\nabla\phi|}\right)\right]\psi\, dx\, dy \\
&\quad + \int_{\partial\Omega}\delta_\varepsilon(\phi)\frac{\nabla\phi}{|\nabla\phi|}\vec{n}\psi\, dx\, dy \\
&= \int_\Omega \delta'_\varepsilon(\phi)|\nabla\phi|\psi\, dx\, dy - \int_\Omega \left[\delta'_\varepsilon(\phi)\frac{|\nabla\phi|^2}{|\nabla\phi|} + \delta_\varepsilon(\phi)\nabla\cdot\left(\frac{\nabla\phi}{|\nabla\phi|}\right)\right]\psi\, dx\, dy \\
&\quad + \int_{\partial\Omega}\delta_\varepsilon(\phi)\frac{\nabla\phi}{|\nabla\phi|}\vec{n}\psi\, dx\, dy \\
&= -\int_\Omega \delta_\varepsilon(\phi)\nabla\cdot\left(\frac{\nabla\phi}{|\nabla\phi|}\right)\psi\, dx\, dy + \int_{\partial\Omega}\delta_\varepsilon(\phi)\frac{\nabla\phi}{|\nabla\phi|}\vec{n}\psi\, dx\, dy,
\end{aligned}$$

da $\nabla(\delta_\varepsilon(\phi)) = \delta'_\varepsilon(\phi)\nabla(\phi)$. Somit muss gelten:

$$\begin{aligned}
-\delta_\varepsilon(\phi)\mathrm{div}\left(\frac{\nabla\phi}{|\nabla\phi|}\right) &= 0 \text{ in } \Omega, \\
\frac{\delta_\varepsilon(\phi)}{|\nabla\phi|}\frac{\partial\phi}{\partial\vec{n}} &= 0 \text{ auf } \partial\Omega,
\end{aligned}$$

wobei $\vec{n}$ die Normale auf dem Rand $\partial\Omega$ und $\partial\phi/\partial\vec{n}$ die Richtungsableitung von ϕ in Richtung $\vec{n}$ ist. Durch Einführung einer künstlichen Zeit $t \in \mathbb{R}_{\geq 0}$ und Verwendung des Gradientenverfahrens findet man somit eine Lösung des Funktionals 2.2 durch Lösen der partiellen Differentialgleichung

$$\begin{aligned}\frac{\partial\phi}{\partial t} = \delta_\varepsilon(\phi)\Bigg[\mu \operatorname{div}\left(\frac{\nabla\phi}{|\nabla\phi|}\right) - \nu - \lambda_1(u_0 - c_1)^2 \\ + \lambda_2(u_0 - c_2)^2\Bigg] = 0 \text{ in } (0,\infty)\times\Omega, \\ \phi(0,x,y) = \phi_0(x,y) \text{ in } \Omega, \\ \frac{\delta_\varepsilon(\phi)}{|\nabla\phi|}\frac{\partial\phi}{\partial\vec{n}} = 0 \text{ auf } \partial\Omega. \end{aligned} \tag{2.3}$$

Die Gleichung 2.3 lässt sich zum Beispiel mithilfe der Finite-Differenzen-Methode numerisch lösen. Für die Segmentierung eines dreidimensionalen Bildes sei $h > 0$ die räumliche Schrittweite, Δt die Zeitschrittweite und $(x_i, y_j, z_k) = (ih, jh, kh)$ für $i = 1,\dots L$, $j = 1,\dots M$ und $k = 1,\dots N$ ein Diskretisierungspunkt. Für die Differenzenquotienten

$$\Delta_x^- \phi := \frac{\phi_{i,j,k} - \phi_{i-1,j,k}}{h}, \tag{2.4}$$

$$\Delta_x^+ \phi := \frac{\phi_{i+1,j,k} - \phi_{i,j,k}}{h}, \tag{2.5}$$

$$\Delta_x^0 \phi := \frac{\Delta_x^+ + \Delta_x^-}{2} = \frac{\phi_{i+1,j,k} - \phi_{i-1,j,k}}{2h} \tag{2.6}$$

gilt auf dem Raumgitter $\Omega = \{0,\dots,L\}\times\{0,\dots,M\}\times\{0,\dots,N\}$ für $h = 1$:

$$\Delta_x^- \phi_{i,j,k} = \phi_{i,j,k} - \phi_{i-1,j,k}, \tag{2.7}$$

$$\Delta_x^+ \phi_{i,j,k} = \phi_{i+1,j,k} - \phi_{i,j,k}, \tag{2.8}$$

$$\Delta_x^0 \phi_{i,j,k} = \frac{\Delta_x^+ + \Delta_x^-}{2} = \frac{\phi_{i+1,j,k} - \phi_{i-1,j,k}}{2}. \tag{2.9}$$

Analog ergeben sich für y, z die entsprechenden Differenzenquotienten. Der Krümmungsterm in der Gleichung 2.3 lässt sich mithilfe der Differenzenquotienten 2.4-2.6 im Punkt

(x_i, y_j, z_k) wie folgt approximieren:

$$\begin{aligned}
\operatorname{div}\left(\frac{\nabla\phi}{|\nabla\phi|}\right) &= \partial_x\left(\frac{\partial_x\phi}{\sqrt{(\partial_x\phi)^2+(\partial_y\phi)^2+(\partial_z\phi)^2}}\right) \\
&\quad + \partial_y\left(\frac{\partial_y\phi}{\sqrt{(\partial_x\phi)^2+(\partial_y\phi)^2+(\partial_z\phi)^2}}\right) \\
&\quad + \partial_z\left(\frac{\partial_z\phi}{\sqrt{(\partial_x\phi)^2+(\partial_y\phi)^2+(\partial_z\phi)^2}}\right) \\
&\approx \Delta_x^- \frac{\Delta_x^+\phi}{\sqrt{(\Delta_x^+\phi)^2+(\Delta_y^0\phi)^2+(\Delta_z^0\phi)^2}} \\
&\quad + \Delta_y^- \frac{\Delta_y^+\phi}{\sqrt{(\Delta_x^0\phi)^2+(\Delta_y^+\phi)^2+(\Delta_z^0\phi)^2}} \\
&\quad + \Delta_z^- \frac{\Delta_z^+\phi}{\sqrt{(\Delta_x^0\phi)^2+(\Delta_y^0\phi)^2+(\Delta_z^+\phi)^2}}.
\end{aligned}$$

Für

$$\begin{aligned}
A_{i,j,k} &:= \frac{\mu}{\sqrt{(\Delta_x^+\phi_{i,j,k})^2+(\Delta_y^0\phi_{i,j,k})^2+(\Delta_z^0\phi_{i,j,k})^2}}, \\
B_{i,j,k} &:= \frac{\mu}{\sqrt{(\Delta_x^0\phi_{i,j,k})^2+(\Delta_y^+\phi_{i,j,k})^2+(\Delta_z^0\phi_{i,j,k})^2}}, \\
C_{i,j,k} &:= \frac{\mu}{\sqrt{(\Delta_x^0\phi_{i,j,k})^2+(\Delta_y^0\phi_{i,j,k})^2+(\Delta_z^+\phi_{i,j,k})^2}},
\end{aligned}$$

lautet somit die diskrete Darstellung der Gleichung 2.3:

$$\begin{aligned}
\frac{\partial\phi_{i,j,k}}{\partial t} = \delta_\varepsilon(\phi_{i,j,k})\Big[&(A_{i,j,k}(\phi_{i+1,j,k}-\phi_{i,j,k}) - A_{i-1,j,k}(\phi_{i,j,k}-\phi_{i-1,j,k})) \\
&+ (B_{i,j,k}(\phi_{i,j+1,k}-\phi_{i,j,k}) - B_{i,j-1,k}(\phi_{i,j,k}-\phi_{i,j-1,k})) \\
&+ (C_{i,j,k}(\phi_{i,j,k+1}-\phi_{i,j,k}) - C_{i,j,k-1}(\phi_{i,j,k}-\phi_{i,j,k-1})) \\
&- \nu - \lambda_1(u_{0,i,j,k}-c_1)^2 + \lambda_2(u_{0,i,j,k}-c_2)^2\Big],
\end{aligned} \tag{2.10}$$

mit $i = 1, \dots L$, $j = 1, \dots M$ und $k = 1, \dots N$. Sei nun $\phi^n_{i,j,k} = \phi(n\Delta t, x_i, y_j, z_k)$ mit $n \geq 0$ und $\phi^0 = \phi_0$, dann liefert das explizite Euler-Verfahren durch

$$\begin{aligned}\frac{\phi^{n+1}_{i,j,k} - \phi^n_{i,j,k}}{\Delta t} = \delta_\varepsilon(\phi^n_{i,j,k}) \Big[& (A^n_{i,j,k}(\phi^n_{i+1,j,k} - \phi^n_{i,j,k}) - A^n_{i-1,j,k}(\phi^n_{i,j,k} - \phi^n_{i-1,j,k})) \\ & + (B^n_{i,j,k}(\phi^n_{i,j+1,k} - \phi^n_{i,j,k}) - B^n_{i,j-1,k}(\phi^n_{i,j,k} - \phi^n_{i,j-1,k})) \\ & + (C^n_{i,j,k}(\phi^n_{i,j,k+1} - \phi^n_{i,j,k}) - C^n_{i,j,k-1}(\phi^n_{i,j,k} - \phi^n_{i,j,k-1})) \\ & - \nu - \lambda_1(u_{0,i,j,k} - c_1(\phi^n))^2 + \lambda_2(u_{0,i,j,k} - c_2(\phi^n))^2 \Big] \end{aligned} \tag{2.11}$$

eine approximative Lösung der Gleichung 2.3. Dabei werden die Werte c_1 und c_2 nach jedem Zeitschritt Δt neu bestimmt und die Randbedingung wird durch Duplizieren der äußersten Gitterpunkte, zum Beispiel durch $\phi_{0,j,k} = \phi_{1,j,k}$, erfüllt. Für das semi-implizite Gauß-Seidel-Verfahren müssen die Werte $\phi_{i-1,j,k}$, $\phi_{i,j-1,k}$ und $\phi_{i,j,k-1}$ auf der rechten Seite der Gleichung 2.11 jeweils für den Zeitschritt $n+1$, alle anderen Werte für den Zeitschritt n ausgewertet werden. Das hierfür notwendige Durchschreiten des Gitters von links nach rechts, von unten nach oben und von der ersten zur letzten Schicht, führt jedoch zu Abhängigkeiten bei der Berechnung der einzelnen Gitterpunkte und reduziert damit die Effizienz einer parallel durchgeführten Berechnung der Gitterpunkte auf einem Multicore-Prozessor, wie zum Beispiel einer Grafikkarte. Optional kann die Kontur nach einer beliebigen Anzahl von Schritten durch die vorzeichenbehaftete Abstandsfunktion

$$\begin{aligned}\psi_r &= \text{sign}(\phi(t))(1 - |\nabla\psi|), \\ \psi(0, \cdot) &= \phi(t, \cdot),\end{aligned}$$

reinitialisiert werden.

2.2 Random Walker

Sei $\Omega \subset \mathbb{R}^d$ ein beschränktes Gebiet und $\partial\Omega = \partial\Omega_1 \cup \partial\Omega_2$ der Rand von Ω, dann entspricht die Lösung $\Phi(x)$ des Dirichlet-Problems

$$\begin{aligned}\Delta\Phi &= 0 \text{ in } \Omega, \\ \Phi &= 1 \text{ auf } \partial\Omega_1, \\ \Phi &= 0 \text{ auf } \partial\Omega_2,\end{aligned} \tag{2.12}$$

der Wahrscheinlichkeit, dass ein an der Stelle $x \in \Omega$ gestarteter Random Walk zuerst den Randbereich $\partial\Omega_1$ trifft bevor er $\partial\Omega_2$ erreicht (Abb. 3). Bei einer Diskretisierung des Gebiets durch ein dreidimensionales Grid zeigt sich dieser Zusammenhang im folgenden Satz:

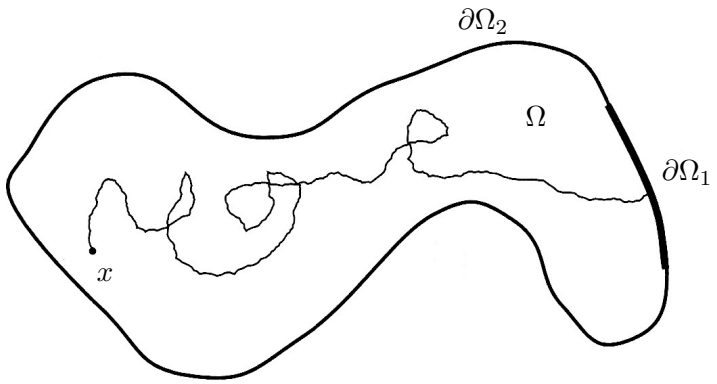

Abbildung 3: Ein Random Walk startet in einem Gebiet Ω und trifft den Rand im Bereich $\partial\Omega_1$ [Evans, 2013].

Satz 2.1. *Für eine Vielzahl von Random Walks, die an der Stelle x starten, wird die Abfolge $\{V_i\}$ der zuerst getroffenen Randwerte notiert. Dann gilt:*

$$\Phi = \lim_{N\to\infty} \left[\frac{1}{N} \sum_{i=1}^{N} V_i \right].$$

Das heißt, die Lösung Φ der Gleichung 2.12 stimmt mit dem Erwartungswert der getroffenen Randwerte überein.

Beweis. Sei (l, m, n) ein beliebiger Punkt, (i, j, k) ein Punkt im Inneren und (r, s, t) ein Punkt auf dem Rand. Sei $P^{r,s,t}_{i,j,k}$ die Wahrscheinlichkeit, dass ein an der Stelle (i, j, k) gestarteter Random Walk den Rand erstmals an der Stelle (r, s, t) trifft. Sei $V_{r,s,t}$ der Wert des Randes an der Stelle (r, s, t) und $E_{l,m,n}$ der Erwartungswert des getroffenen Randwerts eines Random Walks der in (l, m, n) startet. Die Wahrscheinlichkeit für einen Random Walk in (i, j, k) zu starten und in (r, s, t) absorbiert zu werden, kann auf 6 verschiedene Arten erfolgen. Der Random Walk kann sich bei seinem ersten Schritt jeweils mit gleicher Wahrscheinlichkeit in eine der 6 Raumrichtungen bewegen. Das heißt:

$$P^{r,s,t}_{i,j,k} = \frac{1}{6}P^{r,s,t}_{i+1,j,k} + \frac{1}{6}P^{r,s,t}_{i-1,j,k} + \frac{1}{6}P^{r,s,t}_{i,j+1,k} + \frac{1}{6}P^{r,s,t}_{i,j-1,k} + \frac{1}{6}P^{r,s,t}_{i,j,k+1} + \frac{1}{6}P^{r,s,t}_{i,j,k-1}.$$

Durch beidseitige Multiplikation des in (r, s, t) angenommenen Wertes $V_{r,s,t}$ und Aufsummierung aller möglichen Randpunkte, erhält man

$$\begin{aligned}\sum_{r,s,t} V_{r,s,t}P^{r,s,t}_{i,j,k} =& \frac{1}{6}\sum_{r,s,t} V_{r,s,t}P^{r,s,t}_{i+1,j,k} + \frac{1}{6}\sum_{r,s,t} V_{r,s,t}P^{r,s,t}_{i-1,j,k} \\ &+ \frac{1}{6}\sum_{r,s,t} V_{r,s,t}P^{r,s,t}_{i,j+1,k} + \frac{1}{6}\sum_{r,s,t} V_{r,s,t}P^{r,s,t}_{i,j-1,k} \\ &+ \frac{1}{6}\sum_{r,s,t} V_{r,s,t}P^{r,s,t}_{i,j,k+1} + \frac{1}{6}\sum_{r,s,t} V_{r,s,t}P^{r,s,t}_{i,j,k-1}.\end{aligned}$$

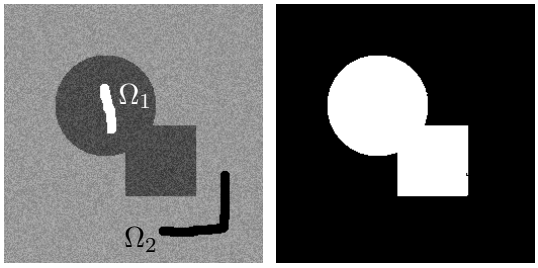

Abbildung 4: Segmentierung basierend auf dem Random Walker Algorithmus. Zweidimensionales Bild mit vorsegmentierten Regionen Ω_1 und Ω_2 (**links**). Resultierende Segmentierung (**rechts**).

Folglich gilt für den Erwartungswert

$$E_{i,j,k} = \sum_{r,s,t} V_{r,s,t} P^{r,s,t}_{i,j,k}$$

der Zusammenhang

$$\begin{aligned} E_{i,j,k} &= \frac{1}{6}E_{i+1,j,k} + \frac{1}{6}E_{i-1,j,k} + \frac{1}{6}E_{i,j+1,k} + \frac{1}{6}E_{i,j-1,k} + \frac{1}{6}E_{i,j,k+1} + \frac{1}{6}E_{i,j,k-1}, \\ E_{r,s,t} &= V_{r,s,t}. \end{aligned}$$

□

Die Verwendung des Dirichlet-Problems zur Segmentierung mehrdimensionaler Bilddaten $u\colon \Omega \subset \mathbb{R}^d \to \mathbb{R}$ ist in der Literatur bekannt als *Random Walker Algorithmus* (RW) zum Lösen des kombinatorischen Dirichlet-Problems [Grady, 2006]. Dabei ergeben sich die Positionen der Ränder durch die von den Benutzern/-innen vorgegebenen Markierungen (Abb. 4). Analog zum Dirichlet-Problem 2.12 wird für jedes Pixel die Wahrscheinlichkeit bestimmt, dass ein dort gestarteter Random Walk in einem anisotropen Medium (dem Bild), zuerst eine der beiden Markierungen erreicht. Die Segmentierung erfolgt dann, indem jedes Pixel derjenigen Region zugeordnet wird, für die die Wahrscheinlichkeit zuerst getroffen zu werden, am größten ist. Die Anisotropie, das heißt die Gewichtung der Random Walks, basiert dabei auf dem Gradienten des Bildes. Gesucht sei also eine harmonische Funktion Φ in einem Gebiet $\Omega \subset \mathbb{R}^d$, die das Dirichlet-Problem

$$\begin{aligned} \nabla \cdot (D \nabla \Phi) = 0, &\text{ in } \Omega, \\ \Phi = 1, &\text{ auf } \partial\Omega_1, \\ \Phi = 0, &\text{ auf } \partial\Omega_2, \end{aligned} \tag{2.13}$$

löst, wobei die Diagonalmatrix

$$D_{i,j} := \begin{cases} \exp(-\beta \partial_{x_i} u), & \text{falls } i = j, \\ 0, & \text{sonst}, \end{cases}$$

mit $D \in \mathbb{R}^{d\times d}$ für ein $\beta > 0$ den anisotropen Diffusionskoeffizienten beschreibt. Hierbei muss also eine elliptische partielle Differentialgleichung zweiter Ordnung, die anisotrope Laplace-Gleichung, unter Einhaltung zweier Dirichlet-Nebenbedingungen gelöst werden. Historisch wurde das Verfahren aus der diskreten Potentialtheorie abgeleitet. Analog zu dem dort definierten Problem elektrischer Schaltkreise ist der RW-Algorithmus im diskreten Raum, das heißt auf einem Graphen, formuliert und erfordert das Lösen eines linearen Gleichungssystems mit einer dünnbesetzten, symmetrischen und positiv definiten Matrix. Durch die Differenzenquotienten 2.7-2.9 lässt sich die partielle Differentialgleichung 2.13 im dreidimensionalen Fall wie folgt approximieren:

$$\begin{aligned}\nabla \cdot (D\nabla\Phi) &= \nabla \cdot \left(\begin{pmatrix} \exp\left(-\beta(\partial_x u)^2\right) & 0 & 0 \\ 0 & \exp\left(-\beta(\partial_y u)^2\right) & 0 \\ 0 & 0 & \exp\left(-\beta(\partial_z u)^2\right) \end{pmatrix} \begin{pmatrix} \partial_x\Phi \\ \partial_y\Phi \\ \partial_z\Phi \end{pmatrix} \right) \\ &\approx \Delta_x^- \left(w_x \Delta_x^+ \Phi\right) + \Delta_y^- \left(w_y \Delta_y^+ \Phi\right) + \Delta_z^- \left(w_z \Delta_z^+ \Phi\right), \end{aligned} \tag{2.14}$$

wobei

$$\begin{aligned} w_x &= \exp\left(-\beta(\Delta_x^+ u)^2\right), \\ w_y &= \exp\left(-\beta(\Delta_y^+ u)^2\right), \\ w_z &= \exp\left(-\beta(\Delta_z^+ u)^2\right), \end{aligned}$$

für einen freien Parameter $\beta > 0$ die ortsabhängige Diffusionsstärke in die jeweilige Raumrichtung beschreiben. Je mehr sich die zwei benachbarten Pixel voneinander unterscheiden, desto schwächer ist die Diffusion, bzw. desto geringer ist die Wahrscheinlichkeit, dass sich ein Random Walk in diese Richtung bewegt. Mithilfe der Approximation 2.14 lässt sich die Gleichung 2.13 nun auf dem Gitter $\Omega_G = \{0,\dots,L\} \times \{0,\dots,M\} \times \{0,\dots,N\}$ wie folgt diskret formulieren:

$$\begin{aligned} 0 =& w_{x_{i,j,k}}(\Phi_{i+1,j,k} - \Phi_{i,j,k}) - w_{x_{i-1,j,k}}(\Phi_{i,j,k} - \Phi_{i-1,j,k}) \\ &+ w_{y_{i,j,k}}(\Phi_{i,j+1,k} - \Phi_{i,j,k}) - w_{y_{i,j-1,k}}(\Phi_{i,j,k} - \Phi_{i,j-1,k}) \\ &+ w_{z_{i,j,k}}(\Phi_{i,j,k+1} - \Phi_{i,j,k}) - w_{z_{i,j,k-1}}(\Phi_{i,j,k} - \Phi_{i,j,k-1}) \\ =& w_{x_{i,j,k}}\Phi_{i+1,j,k} + w_{x_{i-1,j,k}}\Phi_{i-1,j,k} \\ &+ w_{y_{i,j,k}}\Phi_{i,j+1,k} + w_{y_{i,j-1,k}}\Phi_{i,j-1,k} \\ &+ w_{z_{i,j,k}}\Phi_{i,j,k+1} + w_{z_{i,j,k-1}}\Phi_{i,j,k-1} \\ &- (w_{x_{i,j,k}} + w_{x_{i-1,j,k}} + w_{y_{i,j,k}} + w_{y_{i,j-1,k}} + w_{z_{i,j,k}} + w_{z_{i,j,k-1}})\Phi_{i,j,k}, \end{aligned} \tag{2.15}$$

für $i = 1,\dots L$, $j = 1,\dots M$ und $k = 1,\dots N$. Für die Knoten v_l, mit $l = 1,\dots,K$ mit $K = L \cdot M \cdot N$, des Gitters Ω_G mit den entsprechenden Werten $\Phi_{i,j,k}$ ergibt sich aus den Gleichungen 2.15 das lineare Gleichungssystem

$$L\Phi = 0, \tag{2.16}$$

mit der dünnbesetzten, symmetrischen und positiv definiten $K \times K$-Matrix

$$L_{l,m} = \begin{cases} d_l, & \text{falls } l = m, \\ -w_{lm} & \text{falls } v_l \text{ ein Nachbar von } v_m \text{ ist}, \\ 0 & \text{sonst}, \end{cases}$$

wobei $w_{lm} = \exp(-\beta(u_l - u_m)^2)$ und $d_l = \sum_{l \sim m} w_{lm}$. Die Menge der Werte $\Phi_{i,j,k}$ lässt sich nun aufteilen in die Menge der unbekannten Werte Φ_U, das heißt alle Knoten bzw. Pixel, die nicht in den markierten Gebieten Ω_1 und Ω_2 liegen sowie in die Menge der markierten Pixel Φ_M. Ohne Beschränkung der Allgemeinheit werden die Gleichungen des Gleichungssystems 2.16 so umsortiert, dass sich alle bekannten Werte Φ_M oben und alle unbekannten Werte Φ_U unten befinden. Somit gilt:

$$\begin{pmatrix} L_M & B \\ B^T & L_U \end{pmatrix} \begin{pmatrix} \Phi_M \\ \Phi_U \end{pmatrix} = 0.$$

Die Lösung des dadurch entstehenden linearen Gleichungssystems

$$L_U \Phi_U = -B^T \Phi_M$$

liefert dann die gesuchte Wahrscheinlichkeitsverteilung. Das Gleichungssystem kann beispielsweise mit einer geeigneten Vorkonditionierung, wie einem algebraischen Mehrgitterverfahren, und dem GMRES-Verfahren numerisch gelöst werden. Die Methode kann auf mehrere Regionen $\Omega_1, \ldots \Omega_n$ erweitert werden. Dabei müssen insgesamt $n-1$ partielle Differentialgleichungen gelöst werden. Um für jedes x die Wahrscheinlichkeit zu berechnen, dass ein Random Walk zuerst den Rand $\partial\Omega_i$ der Region Ω_i trifft, bevor er den Rand einer der anderen Markierungen trifft, müssen die Nebenbedingungen wie folgt gewählt werden:

$$\begin{aligned} \Phi &= 0 \quad \text{auf } \partial\Omega_1, \\ &\vdots \qquad\quad \vdots \\ \Phi &= 0 \quad \text{auf } \partial\Omega_{i-1}, \\ \Phi &= 1 \quad \text{auf } \partial\Omega_i, \\ \Phi &= 0 \quad \text{auf } \partial\Omega_{i+1}, \\ &\vdots \qquad\quad \vdots \\ \Phi &= 0 \quad \text{auf } \partial\Omega_n. \end{aligned}$$

Dabei müssen insgesamt $n-1$ Gleichungssysteme gelöst werden, da die Summe der Wahrscheinlichkeiten für jedes Pixel 1 ergibt.

2.3 Graph Cut

Beim Graph Cut Verfahren [Boykov and Funka-Lea, 2006] wird ein auf dem Bild konstruierter Graph (Abb. 5b), basierend auf benutzerdefinierten Markierungen (Saatpunkte) $\mathcal{O}$

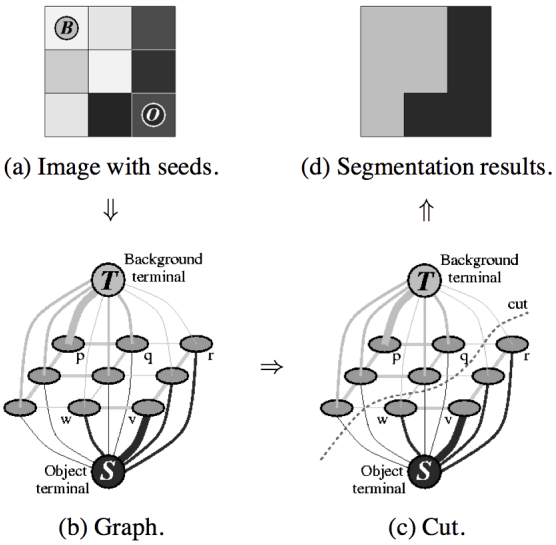

Abbildung 5: Graph mit Terminals für ein 3×3-Bild. a 3×3 Graustufenbild mit Saatpunkten $\mathcal{O} = \{v\}$ und $\mathcal{B} = \{p\}$. **b** Die Dicke der Kanten entspricht den jeweiligen Kosten. **c** Der minimale Schnitt ("cut") unterteilt den Graphen in zwei disjunkte Teilmengen S und T. **d** Segmentierung des Bildes auf Basis des minimalen Schnittes [Boykov and Funka-Lea, 2006].

für das Objekt und $\mathcal{B}$ für den Hintergrund (Abb. 5a), so in zwei Teilmengen S und T (Abb. 5d) unterteilt, dass die Trennlinie (der Schnitt durch das Bild, Abb. 5c) ein vorgegebenes Funktional minimiert. Hierfür werden zu den sogenannten Terminals S und T zwei zusätzliche Knoten eingeführt. Die sogenannte Quelle $s \in S$ und die sogenannte Senke $t \in T$. Der gesuchte minimale Schnitt lässt sich nun anhand des maximalen Flusses von s nach t bestimmen. Im Folgenden sei ein gerichteter Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ bestehend aus den Knoten

$$\mathcal{V} = \mathcal{P} \cup \{S, T\}$$

und den Kanten $\mathcal{E}$, wobei $\mathcal{P}$ die Menge der zu den Knoten korrespondierenden Pixel $p \in \mathcal{P}$ ist. Allen Kanten $e \in \mathcal{E}$ wird ein Gewicht $w(e)$ zugeordnet. Im Folgenden wird $w(e)$ auch häufig Kostenbetrag oder Kapazität genannt. Die Kosten einer gerichteten Kante $\{p, q\}$ können sich dabei von der entgegengesetzten Kante $\{q, p\}$ unterscheiden. Es gibt zwei Kantentypen: sogenannte n-Verbindungen verknüpfen benachbarte Knoten und spiegeln die lokale Umgebung der Knoten wider, sogenannte t-Verbindungen verknüpfen die Knoten mit den Terminals S und T. Jedes Pixel p hat zwei t-Verbindungen, $\{p, S\}$ und $\{p, T\}$. Somit gilt:

$$\mathcal{E} = \mathcal{N} \bigcup_{p \in \mathcal{P}} \{\{p, S\}, \{p, T\}\},$$

wobei $\mathcal{N}$ die Menge aller n-Verbindungen ist. Für die Gewichte der Kanten gilt:

$$w(e) = \begin{cases} B_{p,q}, & \text{falls } e = \{p,q\} \in \mathcal{N}, \\ \lambda \cdot R_p(\text{“hin”}), & \text{falls } e = \{p,S\}, p \notin \mathcal{O} \cup \mathcal{B}, \\ K, & \text{falls } e = \{p,S\}, p \in \mathcal{O}, \\ 0, & \text{falls } e = \{p,S\}, p \in \mathcal{B}, \\ \lambda \cdot R_p(\text{“obj”}), & \text{falls } e = \{p,T\}, p \notin \mathcal{O} \cup \mathcal{B}, \\ 0, & \text{falls } e = \{p,T\}, p \in \mathcal{O}, \\ K, & \text{falls } e = \{p,T\}, p \in \mathcal{B}, \end{cases} \tag{2.17}$$

wobei die regionenbasierten Terme

$$R_p(\text{“obj”}) = -\ln P(\mathcal{I}_p|\text{“obj”}),$$
$$R_p(\text{“hin”}) = -\ln P(\mathcal{I}_p|\text{“hin”}),$$

jeweils dann sehr klein werden, wenn aufgrund der Grauwertverteilungen von $\mathcal{O}$ und $\mathcal{B}$ die Wahrscheinlichkeit sehr groß wird, dass ein Pixel p dem Objekt bzw. dem Hintergrund zugeordnet werden kann. Des Weiteren soll der Kantenterm

$$B_{p,q} = \exp\left(-\frac{(\mathcal{I}_p - \mathcal{I}_q)^2}{2\sigma^2}\right), \tag{2.18}$$

mit dem manuell zu bestimmenden Parameter σ, eine Diskontinuität des Objekts verhindern und ist daher groß, wenn die Intensitäten zweier benachbarter Pixel p und q einander ähnlich sind und verschwindet, wenn sie sich stark unterscheiden. Der Term

$$K = 1 + \max_{p \in \mathcal{P}} \sum_{\{p,q\} \in \mathcal{N}} B_{p,q}$$

soll die richtige Zuordnung der vorsegmentierten Pixel $p \in \mathcal{O}$ bzw. $p \in \mathcal{B}$ garantieren. Ein s-t-Schnitt (oder einfach nur Schnitt) ist eine Teilmenge $C \subset \mathcal{E}$. Ein Schnitt separiert den Graphen in zwei disjunkte Teilmengen S und T, so dass $s \in S$ und $t \in T$. Die Kosten eines Schnittes $C = \{S,T\}$ sind definiert als die Kosten der Kanten. Das heißt:

$$|C| = \sum_{e \in C} w(e).$$

Gesucht sei nun ein Schnitt, der eine optimale Segmentierung liefert. Optimal bedeutet hierbei, dass die Segmentierung $A = (A_1, \ldots, A_p, \ldots, A_{|\mathcal{P}|})$ mit

$$A_p = \begin{cases} \text{“obj”}, & \text{falls } p \text{ dem Objekt zugeordnet wird,} \\ \text{“hin”}, & \text{falls } p \text{ dem Hintergrund zugeordnet wird,} \end{cases}$$

die Kostenfunktion

$$E(A) = \lambda \cdot \sum_{p \in \mathcal{P}} R_p(A_p) + \sum_{\{p,q\} \in \mathcal{N}} B_{p,q} \cdot \delta_{A_p \neq A_q}, \tag{2.19}$$

mit

$$\delta_{A_p \neq A_q} = \begin{cases} 1 & \text{falls } A_p \neq A_q, \\ 0 & \text{falls } A_p = A_q, \end{cases}$$

für einen manuell zu wählenden Modellparameter $\lambda \geq 0$ minimiert. Ziel ist es also eine Segmentierung A zu finden, die die Kostenfunktion 2.19 unter Berücksichtigung der Nebenbedingungen

$$\forall p \in \mathcal{O} : A_p = \text{“obj”}, \tag{2.20}$$
$$\forall p \in \mathcal{B} : A_p = \text{“hin”}, \tag{2.21}$$

minimiert. Jedem realisierbaren Schnitt C lässt sich dabei eine eindeutige, korrespondierende Segmentierung

$$A_p(C) = \begin{cases} \text{“obj”}, & \text{falls } \{p, T\} \in C, \\ \text{“hin”}, & \text{falls } \{p, S\} \in C, \end{cases}$$

zuordnen. Der minimale Schnitt $\widehat{C} = \min_C |C|$ erfüllt folgende Bedingungen [Boykov and Funka-Lea, 2006, S. 121]:

1. C liefert genau eine t-Verbindung für jedes Pixel p.

2. Die Kante $\{p, q\}$ ist in C genau dann, wenn p und q zu jeweils unterschiedlichen Terminals eine t-Verbindungen besitzen.

3. Falls $p \in \mathcal{O}$, dann $\{p, T\} \in C$.

4. Falls $p \in \mathcal{B}$, dann $\{p, S\} \in C$.

Satz 2.2. *Die durch den minimalen Schnitt $\widehat{C}$ definierte Segmentierung $\widehat{A} = A(\widehat{C})$ minimiert das Energiefunktional $E(A)$ 2.19 unter allen Segmentierungen, die die Nebenbedingungen 2.20 & 2.21 erfüllen.*

Beweis. Aus der Definition der Gewichte 2.17 folgt:

$$\begin{aligned} |\widehat{C}| = \min_C |C| &= \min_C \left(\sum_{e \in C} w(e) \right) \\ &= \min_C \left(\sum_{p \notin \mathcal{O} \cup \mathcal{B}} \lambda \cdot R_p(A_p(C)) + \sum_{\{p,q\} \in \mathcal{N}} B_{p,q} \cdot \delta_{A_p(C) \neq A_q(C)} \right) \\ &= \min_C \left(E(A(C)) - \sum_{p \in \mathcal{O}} \lambda \cdot R_p(\text{“obj”}) - \sum_{p \in \mathcal{B}} \lambda \cdot R_p(\text{“hin”}) \right) \\ &= \min_C E(A(C)) - \text{konst} = \min_A E(A) - \text{konst}. \end{aligned}$$

□

Das *Max-Flow-Min-Cut-Theorem* von Ford und Fulkerson [Ford and Fulkerson, 1956] besagt, dass der *maximale Fluss* im Netzwerk vom Quellknoten s zur Senke t genau dem Wert des minimalen Schnitts entspricht. Dabei gibt es nicht notwendigerweise nur einen minimalen Schnitt, sondern es können mehrere Schnitte mit dem gleichen Wert existieren. Bildlich gesprochen entspricht der *maximale Fluss* der maximal möglichen Wassermenge, die durch gerichtete Rohre mit unterschiedlicher maximaler Kapazität von der Quelle zur Senke fließen kann. Diesem Bild folgend, repräsentiert der minimale Schnitt den *Bottleneck*, durch den der maximale Fluss in einem voll ausgelasteten Netzwerk begrenzt wird. Um also den Schnitt zu finden, der die Kostenfunktion 2.19 minimiert, kann entsprechend der maximale Fluss des Netzwerks bestimmt werden. Abgesehen von der Quelle s und der Senke t muss in jeden Knoten genau so viel hineinfließen, wie herausfließt. Das heißt, für einen Fluss $f\colon \mathcal{E} \to \mathbb{R}_{\geq 0}$ mit $f(e) \leq w(e)\ \forall\ e \in \mathcal{E}$ gilt:

$$\forall v \in \mathcal{V}\backslash\{s,t\} : \sum_{e\in\delta^-(v)} f(e) = \sum_{e\in\delta^+(v)} f(e),$$

wobei

$$\delta^-(v) \coloneqq \{e = \{x,v\} \in \mathcal{E} \mid x \in \mathcal{V}\}$$

die Menge der in v hineinführenden und

$$\delta^+(v) \coloneqq \{e = \{v,x\} \in \mathcal{E} \mid x \in \mathcal{V}\}$$

die Menge der aus v hinausführenden Kanten ist. Der Wert eines s-t-Flusses f ist definiert als der Überschuss im Knoten t oder der Betrag des Überschusses im Knoten s, das heißt:

$$|f| = \sum_{e\in\delta^+(s)} f(e) - \sum_{e\in\delta^-(s)} f(e).$$

Um nun den maximalen Fluss zu finden, wird der Fluss im Netzwerk entlang ungesättigter Pfade solange erhöht, bis mindestens eine Kante des Pfades gesättigt ist (Abb. 6 links). Dieser maximal mögliche zusätzliche Fluss erhöht den Gesamtfluss des Netzwerkes. Der maximale Fluss ist erreicht, wenn jeder Pfad mindestens eine gesättigte (saturierte) Kante enthält, das heißt für eine Kante jedes Pfades gilt $f(e) = w(e)$. Die Informationen über den Fluss werden in einem sogenannten Residualgraphen $\mathcal{G}_f$ gespeichert (Abb. 6 rechts). Die Anzahl der Knoten von $\mathcal{G}_f$ entspricht der des ursprünglichen Graphen $\mathcal{G}$. Die Kapazität einer Kante im Residualnetzwerk entspricht der Restkapazität w_f, um die der Fluss noch erhöht werden kann. Der Residualgraph besteht aus denen von f nicht ausgelasteten Kanten und wird um Rückkanten e^* ergänzt, die angeben, um wie viel der Fluss in $\mathcal{G}$ verringert werden kann:

$$\begin{aligned}
&\forall e = \{p,q\} \in \mathcal{E} : w_f(e) = w(e) - f(e), \text{ falls } f(e) < w(e),\\
&\forall e^* = \{q,p\} \in \mathcal{E}_f\backslash\mathcal{E} : w_f(e^*) = f(e).
\end{aligned}$$

Zu Beginn gibt es keinen Fluss. Das heißt $f = 0$ und die Kapazitäten in $\mathcal{G}_f$ entsprechen den Kapazitäten in $\mathcal{G}$. Ein s-t-Fluss f ist genau dann maximal, wenn es keinen augmentierenden Pfad gibt, das heißt, wenn das Restnetzwerk keinen Pfad von s nach t besitzt.

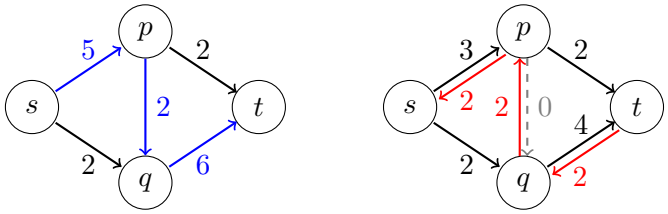

Abbildung 6: links Graph $\mathcal{G}$ mit maximalen Kapazitäten. Der Pfad $W = spqt$ (blau) kann um 2 erhöhten werden. **rechts** Residualgraph $\mathcal{G}_f$ nach Erhöhung des Pfades $W = spqt$ um 2 mit Restkapazitäten w_f (schwarz) und Rückkanten (rot).

Zur Bestimmung des Restflussgraphen kann zum Beispiel der Algorithmus von Ford und Fulkerson verwendet werden [Ford and Fulkerson, 1956]. Das Verfahren terminiert, sobald kein augmentierender Pfad mehr gefunden wird. Sind die Kapazitäten nichtnegative ganze Zahlen und werden die Flüsse in jedem Schritt um 1 erhöht, ist eine Terminierung nach endlich vielen Schritten garantiert. Aus diesem Grund werden die berechneten Kapazitäten 2.17 in ganzzahlige Werte umgerechnet. Der nachfolgende Äquivalenzbeweis für das *Max-Flow-Min-Cut-Theorem* liefert zugleich eine Methode, mit der aus der Berechnung des maximalen Flusses der minimale Schnitt bestimmt werden kann. Standardalgorithmen mit polynomialer Laufzeit, wie der Algorithmus von Dinic, basieren ebenfalls auf der Grundidee der augmentierenden Pfade von Ford und Fulkerson.

Satz 2.3 (*Max-Flow-Min-Cut-Theorem*)**.** *Die folgenden Aussagen sind äquivalent:*

1. *f ist der maximale Fluss in $\mathcal{G}$.*
2. *Das Residualnetzwerk $\mathcal{G}_f$ enthält keinen augmentierenden Pfad.*
3. *Für mindestens einen Schnitt $C = \{S, T\}$ entspricht der Wert des Flusses der Kapazität des Schnittes.*

Beweis. $(1 \Rightarrow 2)$ Angenommen es gibt einen augmentierenden Pfad in $\mathcal{G}_f$, dann gilt für jede Kantenkapazität des Pfades $w_f(e) > 0$ und somit $w(e) > f(e)$. Daraus folgt jedoch, dass der Fluss in $\mathcal{G}$ entlang dieses Pfades vergrößert werden kann und f somit nicht maximal ist, da jeder zusätzliche Fluss den Gesamtfluss vergrößert. $(2 \Rightarrow 3)$ Wenn es keinen augmentierenden Pfad gibt, dann teile den Graphen in zwei disjunkte Teilmengen S und T, wobei S die von der Quelle s im Residualnetzwerk erreichbaren Knoten (inklusive s) und T die restlichen Knoten enthält. Dann gilt $|C| - |f| = 0$ und somit $|f| = |C|$. Entspräche nämlich die Kapazität des Schnittes nicht dem Fluss f, so gäbe es eine Kante $\{x, y\}$ mit $x \in S, y \in T$, die nicht gesättigt ist und somit eine Kante von s nach t in $\mathcal{G}_f$. Dies bedeutet jedoch, dass mindestens ein weiterer Knoten in T erreichbar wäre, denn für jede Teilmenge von Knoten $\mathcal{V}^*$ mit $s \in \mathcal{V}^*$ und $t \notin \mathcal{V}^*$ gilt $|f| = f_{\text{out}}(\mathcal{V}^*) - f_{\text{in}}(\mathcal{V}^*)$. $(3 \Rightarrow 1)$ Angenommen f ist nicht maximal, das heißt, f kann vergrößert werden, dann ist die Kapazität für min-

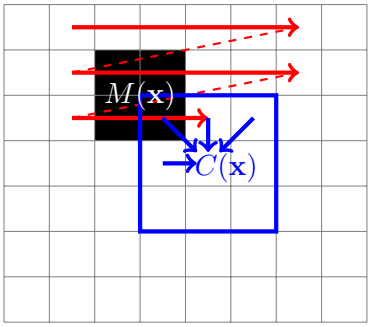

Abbildung 7: Bewegung (rot) des Forwärtskernels (blau) im *Raster-Scan*-Algorithmus zur approximativen Bestimmung der geodätischen Distanz $D(\mathbf{x})$ zur Vorsegmentierung $M(\mathbf{x})$ durch die Funktion $C(\mathbf{x})$.

destens einen Schnitt nicht voll ausgelastet, da für alle Schnitte $|f| \leq |C|$ gilt. Außerdem gilt für keinen Schnitt $|f| = |C|$, da sonst kein augmentierender Pfad existierte und der Fluss somit maximal wäre. □

2.4 Geodätische Distanz

Sei $\Omega \subset \mathbb{R}^d$ beschränkt, $\mathcal{I}\colon \Omega \to \mathbb{R}$ ein gegebenes Bild, Ω_1 eine markierte Region (Saatpunkte) mit der zugehörigen binären Maske (Abb. 7)

$$M(\mathbf{x}) = \begin{cases} 0, & \text{falls } \mathbf{x} \in \Omega_1, \\ 1, & \text{sonst}, \end{cases}$$

dann ist die geodätische Distanz eines Pixel $\mathbf{x}$ zur Markierung Ω_1 definiert als:

$$D(\mathbf{x}; M, \nabla\mathcal{I}) = \min_{\{\mathbf{x}' | M(\mathbf{x}')=0\}} d(\mathbf{x}, \mathbf{x}'),$$

mit

$$d(\mathbf{a}, \mathbf{b}) = \min_{\mathbf{\Gamma} \in \mathcal{P}_{\mathbf{a},\mathbf{b}}} \int_0^1 \sqrt{\|\mathbf{\Gamma}'(s)\|^2 + \gamma^2 (\nabla\mathcal{I}(\mathbf{\Gamma}(s)) \cdot \mathbf{u}(s))^2}\, ds,$$

wobei $\mathcal{P}_{\mathbf{a},\mathbf{b}}$ die Menge aller möglichen Pfade $\mathbf{\Gamma}\colon [0,1] \to \mathbb{R}^d$ zwischen den Punkten $\mathbf{a}$ und $\mathbf{b}$ ist. Ferner ist $\mathbf{u}(s) = \mathbf{\Gamma}'(s)/\|\mathbf{\Gamma}'(s)\|$ der Einheitsvektor tangential zur Richtung des Pfades. Der Faktor γ gewichtet den Einfluss des Bildgradienten auf die Länge der geodätischen Kurve. Für $\gamma = 0$ entspricht D der euklidischen Distanz. Sind nun für n gesuchte Segmente eines Bildes die Saatpunkte $\Omega_1, \ldots, \Omega_n$ gegeben, so lässt sich für jedes Pixel eine Zuordnung bestimmen, indem zunächst die geodätischen Distanzen zu den einzelnen Gebieten Ω_i berechnet werden und dann für das Pixel das Gebiet mit der kürzesten Entfernung gewählt wird [Criminisi et al., 2008]. Eine approximative Lösung von D erhält man beispielsweise durch eine Wellenfront-Entwicklung wie bei den sogenannten *Fast Marching*

Methods [Sethian, 2015, Yatziv et al., 2006] oder durch den *Raster-Scan*-Algorithmus [Toivanen, 1996]. Im zweidimensionalen Fall wird beim *Raster-Scan*-Algorithmus ein Kernel, zum Beispiel der Größe 3×3, zuerst von links oben nach rechts unten bewegt (Abb. 7). Dabei wird die Funktion

$$C(x,y) = \min \begin{cases} C(x-1,y-1) + \sqrt{p_2^2 + \gamma(\mathcal{I}(x-1,y-1) - \mathcal{I}(x,y))^2}, \\ C(x,y-1) + \sqrt{p_1^2 + \gamma(\mathcal{I}(x,y-1) - \mathcal{I}(x,y))^2}, \\ C(x+1,y-1) + \sqrt{p_2^2 + \gamma(\mathcal{I}(x+1,y-1) - \mathcal{I}(x,y))^2}, \\ C(x-1,y) + \sqrt{p_1^2 + \gamma(\mathcal{I}(x-1,y) - \mathcal{I}(x,y))^2}, \\ \nu M(x,y). \end{cases} \tag{2.22}$$

bestimmt, wobei p_1 und p_2 bildunabhängigen Distanzen entsprechen. Üblicherweise werden hierfür die euklidischen Distanzen $p_1 = 1$ und $p_2 = \sqrt{2}$ gewählt. Anschließend erfolgt eine Aktualisierung der Funktion $C(x,y)$ durch Verschiebung eines zum Vorwärtskernel punktsymmetrischen Kernels von links unten nach rechts oben. Das Verfahren lässt sich ohne Weiteres auf höhere Dimensionen erweitern. Die Genauigkeit der Approximation hängt dabei von der Anzahl der Durchführungen und von der Kernelgröße ab.

3 GPU-basierte Random Walks

Im Gegensatz zu den in Kapitel 2 vorgestellten halbautomatischen Segmentierungsverfahren hängt der Segmentierungserfolg, der hier vorgestellten GPU-basierten Methode, nicht von einem freien und manuell zu wählenden Parameter ab [Lösel et al., 2020]. Durch diese Parameterfreiheit konnte ein leicht und intuitiv nutzbares Verfahren entwickelt werden, das bei vielen unterschiedlichen Datensätzen eine hohe Segmentierungsgenauigkeit aufweist, gleichzeitig aber keine Konfiguration durch die Nutzer/-innen erfordert. Wie auch bei der traditionellen und weit verbreiteten Vorgehensweise der linearen Interpolation, werden zunächst einzelne Schichten des 3D-Volumens manuell vorsegmentiert (Abb. 8). Anschließend wird das übrige Volumen vollautomatisch durch speziell gewichtete Random Walks segmentiert. Die gegenseitige Unabhängigkeit der Random Walks ermöglicht eine Berechnung auf massiv parallel arbeitenden Prozessoren. Durch die hardwarespezifische Implementierung des Verfahrens auf Grafikprozessoren können somit sehr große Bilddatensätze in kürzester Zeit verarbeitet werden. Hierfür werden die sogenannten GPU-basierten Random Walks in einer Monte-Carlo-Simulation unter Verwendung mehrerer Grafikprozessoren berechnet. Die Unabhängigkeit der Random Walks ist nicht nur der Schlüssel für eine äußerst effiziente Berechnung, sondern auch für die spezielle parameterfreie Definition der Gewichte. Nach der Vorsegmentierung einzelner Schichten starten von dort etliche Random Walks und bewegen sich durch die Berechnung gewichteter Zufallszahlen durch das Volumen (Abb. 9). Im Laufe der Zeit werden dabei die Voxel des Volumens durch die von verschiedenen Label aus startenden Random Walks getroffen. Die Segmentierung der Bilddaten erfolgt anschließend indem jedes Voxel der Region zugeordnet wird, von der die meisten Treffer ausgingen. Voxel, die nie getroffen wurden, werden automatisch dem Hintergrund zugewiesen. In einem dreidimensionalen Volumen kann jeder Schritt eines Random Walks als zufälliger Wurf auf eine Scheibe mit sechs unterschiedlich großen Feldern betrachtet werden. Die Größe der Felder entspricht dabei jeweils der Wahrscheinlichkeit eines Random Walks, in eine der sechs möglichen Richtungen zu gehen. Je größer ein Feld ist, also je größer das Gewicht, desto mehr entspricht das in einer neuen Richtung gelegene Voxel der Startposition des Random Walks. Die Parallelisierung der Random Walks wird erreicht, indem jedem sogenannten *Thread* ein Pixel in den vorsegmentierten Schichten zugeordnet wird und alle darin startenden Random Walks von diesem *Thread* berechnet werden. Die benötigten Zufallszahlen werden dabei mithilfe des *Multiple Recursive Random Number Generator* (*MRG32k3a*) [L'Ecuyer, 1999] bestimmt. Die Berechnung der Zufallszahlen ist aus Effizienzgründen direkt in die Berechnung der Random Walks integriert. Auf diese Weise wird zusätzlich benötigter Speicherbedarf und Datentransfer zwischen Hauptarbeitsspeicher und Grafikprozessor vermieden.

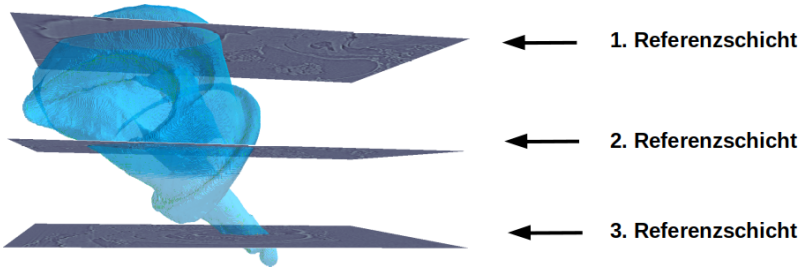

Abbildung 8: Vorsegmentierte Referenzschichten. Die Referenzschichten dienen der Initialisierung der GPU-basierten Random Walks zur Segmentierung eines biologischen Schraubengelenks [van de Kamp et al., 2011].

3.1 Modellierung der GPU-basierten Random Walks

Definition 3.1 (Gewichte). *Sei* $\mathcal{I}\colon \Omega \to \mathbb{R}$ *ein Bild mit* $\Omega \subset \mathbb{R}^3$ *und* $A\colon \Omega \to \{-1, 0, 1, 2 \ldots\}$ *sei gegeben durch die Vorsegmentierung einzelner Schichten, wobei* $A(x) < 0$ *für alle nicht vorsegmentierten Voxel. Dann sind die Gewichte zu einer Stelle* $x_0 \in \Omega$ *mit* $A(x_0) \geq 0$ *definiert als*

$$w_{x_0}(y) = \begin{cases} \exp\left(-\frac{(\mathcal{I}(x_0)-\mathcal{I}(y))^2}{2\sigma_{x_0}^2}\right), & \text{falls } y \in \Omega, \\ 0, & \text{falls } y \notin \Omega, \end{cases} \tag{3.1}$$

wobei

$$\sigma_{x_0}^2 = \max\left(\frac{1}{|M|}\sum_{x\in M}(\mathcal{I}(x_0) - \mathcal{I}(x))^2, 1\right), \tag{3.2}$$

mit $M = \{x \in U \mid A(x) = A(x_0)\}$, *die mittlere quadratische Abweichung der Grauwerte* $\mathcal{I}(x)$ *in einer Umgebung* $U \subset \Omega$ *(Abb. 10) vom Grauwert an der Stelle* x_0 *ist.*

Definition 3.2 (Gewichteter Random Walk). *Für wie zuvor gegebene Gewichte* w_{x_0} *zur Startposition* $x_0 \in \Omega$ *sei* $(Z_1, Z_2, \ldots)$ *eine Folge von Zufallsvariablen mit Werten in der Menge* $\{(1,0,0)^T, (-1,0,0)^T, (0,1,0)^T, (0,-1,0)^T, (0,0,1)^T, (0,0,-1)^T\}$, *deren bedingte Verteilung gegeben ist durch*

$$P_{x_0}\left(\{Z_k = z_i\} \;\middle|\; \{X_{k-1} = x_{k-1}\}\right) = \frac{w_{x_0}(x_{k-1} + z_i)}{\sum_{j=1}^{6} w_{x_0}(x_{k-1} + z_j)}$$

für $k \geq 1$ *und* $i = 1, \ldots, 6$. *Dann heißt der durch*

$$X_n = x_0 + \sum_{k=1}^{n} Z_k, \quad n \in \mathbb{N},$$

definierte stochastische Prozess $(X_n)_{n\in\mathbb{N}}$ *gewichteter Random Walk in* $\mathbb{R}^3$ *und* $n \in \mathbb{N}$ *die Anzahl der durchgeführten Schritte.*

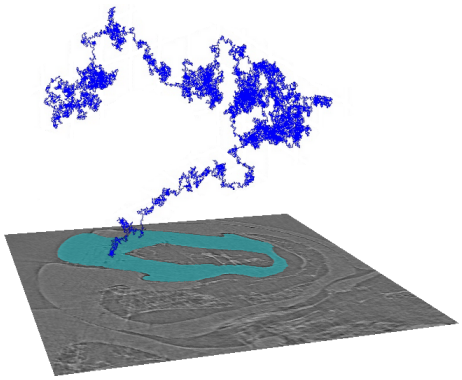

Abbildung 9: Gewichteter GPU-basierter Random Walk. Die Grafik illustriert einen dreidimensionalen Random Walk nachdem er an einer Stelle x_0 in einer vorsegmentierten Schicht gestartet und durch das Bildvolumen gelaufen ist [Lösel and Heuveline, 2016].

Wenn $\sigma_{x_0}^2$ sehr klein ist, dann ist die Wahrscheinlichkeit groß, dass der Random Walk in einer Region bleibt, deren Grauwerte mit dem der Startposition übereinstimmen. Für eine größere Varianz in der Startregion wird $\sigma_{x_0}^2$ zunehmend größer und damit werden, unabhängig von der Startposition, alle Richtungen allmählich gleich wahrscheinlich. Da in dieser Arbeit ausschließlich die gewichtete Form betrachtet wird, wird der Einfachheit halber in der Regel auf die Bezeichnung „gewichtet“ verzichtet.

3.2 Multi-GPU Implementierung

Ein Grafikprozessor (*Graphics Processing Unit*, GPU) ist ein ursprünglich auf die Berechnung von Grafiken spezialisierter und optimierter Prozessor. Da jedes Pixel individuell betrachtet und somit alle Pixel eines Bildes parallel verarbeitet werden können, um zum Beispiel den Helligkeitswert aller Pixel gleichmäßig zu erhöhen oder ein 3D-Modell für eine zweidimensionale Darstellung in ein 2D-Bild zu konvertieren (3D-*Rendering*), kann jede dieser Ausführungen einem sogenannten *Thread* zugeordnet werden. Jeder *Thread* stellt dabei einen sequenziellen Teil des Gesamtprozesses dar. Die einzelnen *Threads* können nun wiederum mehreren physischen Prozessoren zugeordnet und von diesen parallel verarbeitet werden. Moderne GPUs besitzen viele solcher Prozessoren. Die NVIDIA Tesla A100 beispielsweise besitzt je nach Ausbaustufe 108 oder 128 *Shader*-Multiprozessoren. Die hoch parallele Architektur macht GPUs für Algorithmen, die große Datenblöcke parallel verarbeiten können, wesentlich effizienter als sequentiell arbeitende Prozessoren wie die Zentrale Recheneinheit (*Central Processing Unit*, CPU). Durch den ursprünglichen Fokus auf Multimedia-Anwendungen waren die *Shader* sehr stark auf die Berechnung von grafischen Operationen und die hierfür zur Verfügung gestellten Funktionen beschränkt. Erst später entstanden nach und nach frei programmierbare *Shader*, um diese für beliebige Aufgaben, wie wissenschaftliche Simulationen oder Matrixmultiplikationen (Abschnitt 4.2.2), einzusetzen (*General Purpose Computation on Graphics Processing Unit*, GPGPU). Neben dem

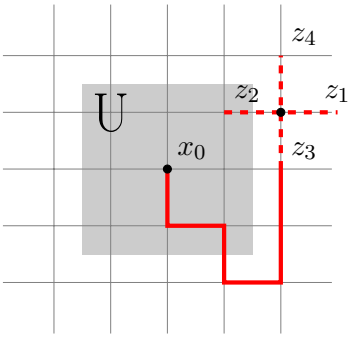

Abbildung 10: GPU-basierter Random Walk mit Startposition x_0 nach sieben gelaufenen Schritten. Hier illustrativ innerhalb der vorsegmentierten Schicht. Die grau eingefärbte lokale Umgebung U dient zur Berechnung der Wahrscheinlichkeiten für die sechs möglichen Laufrichtungen z_i mit $i = 1, \ldots, 6$. Die Laufrichtungen nach oben bzw. unten ($i = 5, 6$) innerhalb des Volumens sind hier nicht dargestellt [Lösel and Heuveline, 2017].

offenen Standard OpenCL ist das von NVIDIA entwickelte CUDA ein weiteres auf der traditionellen Sprache C basierendes *Framework* zur Umsetzung von GPGPU-Berechnungen auf paralleler Hardware, welches jedoch im Gegensatz zu OpenCL ausschließlich in Kombination mit NVIDIA GPUs verwendet werden kann. CUDA besitzt gegenüber C mehrere Erweiterungen. Wesentlich ist hierbei vor allem die Definition eines *Kernels*. Ein *Kernel* wird im Gegensatz zu einer normalen C-Funktion bei seinem Aufruf nicht nur einmal, sondern sehr häufig durch mehrere *Threads* auf der GPU ausgeführt. Möchte man beispielsweise in C jeden Wert eines dreidimensionalen Bildes jeweils um eins erhöhen, benötigt man hierfür eine *for*-Schleife, um über die einzelnen Einträge des Bildes zu iterieren. Auf der GPU entfällt diese *for*-Schleife und wird durch einen *Kernel* ersetzt (Programm 1). Dieser *Kernel* weist die auf jedem Pixel unabhängig arbeitenden Prozesse jeweils einem *Thread* zu.

Programm 1: CUDA-*Kernel* zur parallelen Erhöhung der Pixelwerte eines dreidimensionalen Bildes auf einer GPU.

```
1  __global__ void Funktion(int *img, int n) {
2      int ix = blockIdx.x * blockDim.x + threadIdx.x;
3      int iy = blockIdx.y * blockDim.y + threadIdx.y;
4      int iz = blockIdx.z * blockDim.z + threadIdx.z;
5      int nx = gridDim.x * blockDim.x;
6      int ny = gridDim.y * blockDim.y;
7      uint index = iz * nx * ny + iy * nx + ix;
8      if (index < n) {
9          img[index] += 1;
10         }
11     }
```

Die Verarbeitung erfolgt somit zumindest in der Theorie vollständig parallel. In der Praxis muss jedoch beachtet werden, dass GPUs über eine beschränkte Anzahl physischer Recheneinheiten verfügen und diese Anzahl in der Regel deutlich kleiner ist als die Anzahl der für

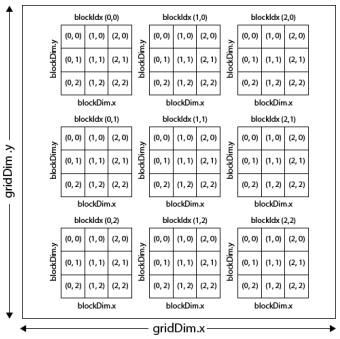

Abbildung 11: *Grid-Block-Thread* Struktur zur Verarbeitung eines CUDA-*Kernels*. Das 3×3-Grid unterteilt sich in 3×3-Blöcke mit jeweils 9 *Threads* [Jeremiah, 2011].

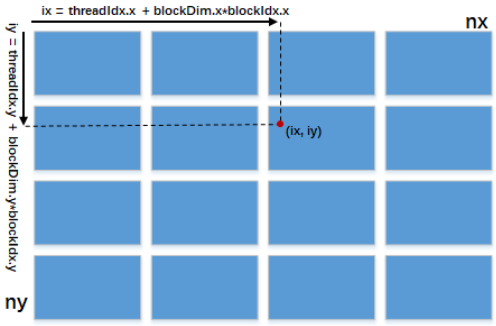

Abbildung 12: Berechnung eines *Thread*-Index in einem 2D-Grid. Zur Bestimmung des Index müssen sowohl die Blockindizes als auch die Griddimensionen berücksichtigt werden [Cheng et al., 2014].

die Berechnung definierten *Threads*. Für die Verarbeitung werden die *Threads* zu 1D-, 2D- oder 3D-Blöcken, bestehend aus jeweils mehreren *Threads*, zusammengefasst. Diese Blöcke wiederum werden zu einem 1D-, 2D- oder 3D-Grid zusammengefasst (Abb. 11). Die Anzahl der *Threads* pro Block und die Anzahl der Blöcke werden vor Aufruf des *Kernels* durch

$$block = (blockDim.x, blockDim.y, blockDim.z),$$
$$grid = (gridDim.x, gridDim.y, gridDim.z),$$

festgelegt. Dies wiederum definiert die Gesamtzahl der gestarteten CUDA-*Threads*, deren Indizes sich über

$$ix = blockIdx.x * blockDim.x + threadIdx.x,$$
$$iy = blockIdx.y * blockDim.y + threadIdx.y,$$
$$iz = blockIdx.z * blockDim.z + threadIdx.z,$$

bestimmen lassen (Programm 1 & Abb. 12). Für die maximalen x, y und z-Dimensionen eines Blocks gilt jeweils 1024, 1024 und 64, wobei die absolute Anzahl der *Threads* pro Block 1024 nicht überschreiten darf. Die *Threads* eines Blocks werden vom gleichen Streamprozessor verarbeitet und können beispielsweise über *Shared Memory* und Synchronisationsbarrieren miteinander kommunizieren, was zum Beispiel bei der Berechnung komplexerer Aufgaben, wie der Matrixmultiplikation (Abschnitt 4.2.2) notwendig ist. Für eine größtmögliche Effizienz sollte die Berechnung der *Threads* jedoch vollständig unabhängig voneinander erfolgen. Das heißt, ein *Thread* sollte bei seiner Berechnung nicht auf das Ergebnis oder Zwischenergebnis eines anderen *Threads* angewiesen sein. Die Blöcke in einem Grid müssen vollständig unabhängig voneinander ausführbar sein, da eine Kommunikation oder Kooperation zwischen Blöcken in einem Grid nicht möglich ist. Lediglich einige atomare Funktionen wie *atomicAdd* erlauben eine wechselseitige Einflussnahme.

3.2.1 GPU-Implementierung der Random Walks

Die in Abschnitt 3.1 beschriebenen Random Walks können vollständig unabhängig voneinander berechnet werden und sind damit ideal geeignet, um durch Grafikprozessoren realisiert zu werden. Um eine parallele Berechnung mit CUDA zu ermöglichen (Programm 2), wird jedem Pixel der vorsegmentierten Schichten ein *Thread* zugewiesen (Zeile 33). Dieser *Thread* übernimmt die Berechnung aller Random Walks, die in diesem Pixel starten. Da die Größe des Blocks $block = (32, 32, 1)$ fest gewählt ist, muss die Größe des Grids

$$grid = ((xsh//32) + 1, (ysh//32) + 1, N),$$

wobei // die ganzzahlige Division beschreibt, an die Anzahl N und die Größe $xsh \times ysh$ der vorsegmentierten Schichten angepasst werden. Dabei wird das Grid so bestimmt, dass es einen Überhang des jeweils letzten Blocks über den Rand der vorsegmentierten Schichten gibt. Da das Grid nicht wie in Programm 1 über das gesamte Bildvolumen, sondern über die vorsegmentierten Schichten definiert ist, stimmen *Thread*-Index und Startposition des Random Walks nicht zwangsläufig überein. Zwar lassen sich die Spalten- und Zeilenindizes sowohl des *Threads* als auch der Startposition eines Random Walks innerhalb einer vorsegmentierten Schicht (*column* in der Zeile 29, *row* in der Zeile 30) wie zuvor in Programm 1 bestimmen, jedoch muss der z-Index der Startposition (*plane* in der Zeile 32) anhand des übergebenen *indices*-Arrays, das jeder vorsegmentierten Schicht ihre z-Position im Volumen zuweist, ermittelt werden. Außerdem lässt sich durch den Überhang die Position eines Pixels nicht mehr wie zuvor über die Griddimensionen bestimmen, sondern muss anhand der zusätzlich übermittelten Bildmaße (xsh, ysh) in der Zeile 33 berechnet werden. Insgesamt werden somit $((xsh//32) + 1) \times ((ysh//32) + 1) \times N$ *Threads* definiert, wovon einige dem Überhang zugeordnet werden. Da hier jedoch keine Bildinformationen vorhanden sind und somit keine Random Walks berechnet werden können, wird in der Zeile 35 zunächst geprüft, ob sich die berechnete Startposition (*column*, *row*, *plane*) des Random Walks im, durch das dreidimensionale Bild vorgegebenen, Volumen befindet. Falls dies nicht der Fall ist, wird der entsprechende *Thread* abgebrochen.

Übergabe von Daten

Neben der Bildgröße und den z-Indizes der vorsegmentierten Schichten, werden dem *Kernel* noch folgende weitere Daten, die bereits vor Aufruf des *Kernels* auf den GPU-Speicher transferiert wurden, in der Zeile 26 übergeben: das in diesem *Kernel*-Aufruf berechnete Label $segment \in \mathbb{N}$ von dem die Random Walks starten sollen, das dreidimensionale Bild raw der Größe $xsh \times ysh \times zsh$ Voxel, die vorsegmentierten Schichten *slices* der Größe $xsh \times ysh \times N$ Voxel, das nach jedem *Kernel*-Aufruf auf Null zurückgesetzte Array a der Größe $xsh \times ysh \times zsh$ Voxel zum Abspeichern der Treffer, die Anzahl der von jedem Random Walk durchgeführten Schritte $sorw \in \mathbb{N}$ und die Anzahl der in jedem Pixel startenden Random Walks $nbrw \in \mathbb{N}$. Der *Kernel* wird für jedes Label einmal aufgerufen, um

die Random Walks, die in diesem Label starten, zu berechnen und die davon ausgehenden Treffer abzuspeichern. Aus diesem Grund werden bei jedem *Kernel*-Aufruf alle zu einem anderen Label gehörenden *Threads* in der Zeile 37 beendet.

Initialisierung und Allokation

In den Zeilen 40-55 wird der Speicher für die Variablen zur Berechnung der Gewichte und der für die Random Walks benötigten Zufallszahlen initialisiert. Als Saatpunkt zur Berechnung der jeweils ersten Zufallszahl dient der *Thread*-Index (Zeile 33). Da die Startposition des Random Walks im 3D-Volumen nicht unbedingt mit dem *Thread*-Index übereinstimmt, muss diese in der Zeile 60 anhand der Werte *column*, *row* und *plane* bestimmt werden. In den Zeilen 57-59 wird die Position der Random Walks initialisiert und in den Zeilen 61 & 62 die Anzahl der gelaufenen Schritte sowie die Anzahl der gestarteten Random Walks.

Einlesen der Bilddaten

In der Zeile 65 wird der Grauwert an der Startposition eingelesen. Da die meisten Bilddaten als ganzzahlige 8-Bit Werte ohne Vorzeichen vorliegen, würde eine vorherige Konvertierung zu einem von CUDA verarbeitbaren 32-Bit Array den Speicherbedarf vervierfachen. Da die Bilddaten zusätzlich sehr große Dimensionen annehmen können (Abschnitt 3.5), wird das als 8-Bit abgespeicherte Bild vom *Kernel* zunächst als 32-Bit Array aufgefasst (Zeile 26), ohne dabei jedoch die abgespeicherten Bytes zu verändern. Um die Daten anschließend richtig einzulesen, erfolgt ein *Pointer Casting* zu dem von CUDA interpretierbaren Datentyp *char*. Das heißt, anstatt fälschlicherweise 4 Bytes (32-Bit) einzulesen, wird nun nur noch 1 Byte (8-Bit) eingelesen und anschließend zur weiteren Verarbeitung in den Datentyp *float* umgewandelt.

Mittlere quadratische Abweichung

In der Zeile 68 wird die mittlere quadratische Abweichung 3.2 der unmittelbar zur Startposition benachbarten Pixel mithilfe der in den Zeilen 1-19 realisierten Funktion bestimmt. Dabei werden nur die Grauwerte berücksichtigt, deren zugehörige Vorsegmentierung mit der der Startposition übereinstimmt. Die Bilddaten werden analog zum vorherigen Abschnitt eingelesen.

Berechnung der Gewichte & Bestimmung der Laufrichtung

In der *while*-Schleife, Zeile 72-147, werden nacheinander alle an der vorgegebenen Startposition startenden Random Walks berechnet. Für jeden Schritt des Random Walks werden in den Zeilen 75-80 und mithilfe der in den Zeilen 21-24 definierten Funktion *weight* die Gewichte 3.1 der 6 möglichen Laufrichtungen bestimmt. In den Zeilen 82-86 wird ein aus 6 Abschnitten bestehendes Intervall konstruiert, dessen Gesamtlänge der Summe aller Gewichte entspricht. Die Länge der einzelnen Abschnitte entspricht dabei jeweils der Größe der 6 Gewichte.
In den Zeilen 91-116 wird eine auf dem Intervall $[0, 1)$ gleichverteilte Zufallszahl mithilfe des *MRG32k3a*-Algorithmus [L'Ecuyer, 1999] berechnet. Die für die Berechnung der jeweils nächsten Zufallszahl benötigten Werte ergeben sich sukzessiv aus den innerhalb des Algorithmus berechneten Werten (Zeilen 96-98 & 106-108). Die somit erzeugte Pseudozufallszahl wird in der Zeile 112 bzw. in der Zeile 115 mit der zuvor berechneten Intervalllänge multipliziert. Je nachdem in welchen der 6 Abschnitte der dadurch berechnete Wert fällt, entscheidet sich, wohin der Random Walk als nächstes läuft (Zeilen 119-124).

Durchführung eines Laufschrittes

Befindet sich die dadurch bestimmte neue Position im Volumen mindestens ein Pixel vom Rand des Bildes entfernt (Zeile 128), bewegt sich der Random Walk zu dieser neuen Position. Hierbei unterscheidet sich die Implementierung von der Definition des gewichteten Random Walks (Definition 3.2). Um nicht in jedem Iterationsschritt für alle potentiellen Laufrichtungen überprüfen zu müssen, ob sich diese noch innerhalb des Volumens befinden, wird die Bewegung zu den äußersten Voxel des Volumens nicht ausgeführt. Die Indizes der Position sowie die Anzahl der durchgeführten Schritte müssen entsprechend aktualisiert werden (Zeilen 129-132 & 136). In der Zeile 133 wird der Treffer des Voxels, das sich an der neuen Position befindet, in das dafür vorgesehenen Array a geschrieben. Da dieses Array aufgrund seiner Größe nur im Hauptspeicher der GPU (*Global Memory*) abgelegt werden kann und alle Random Walks des Segments parallel berechnet und somit potentiell gleichzeitig auf denselben hierfür allokierten Speicherplatz schreiben können, wird die CUDA Funktion *atomicAdd* benötigt, um etwaige Schreibkonflikte verschiedener *Threads* aufzulösen.

Zurücksetzen des Random Walks

Ist die Anzahl maximal möglicher Schritte erreicht (Zeile 139), werden die Indizes der Position auf die Startposition und der Zähler für die Schritte des Random Walks auf Null zurückgesetzt. Anschließend wird der Zähler der insgesamt durchgeführten Random Walks um eins erhöht und ein neuer Random Walk gestartet.

Programm 2: CUDA-*Kernel* für eine parallele Berechnung der GPU-basierten Random Walks zur Segmentierung volumetrischer Bilddaten.

```
1  __device__ float varf(int position,int index,float v,float *raw,int
       segment,int *labels,int xsh) {
2      float tmp;
3      float dev = 0;
4      float summe = 0;
5      for (int n = -1; n < 2; n++) {
6          for (int o = -1; o < 2; o++) {
7              if (labels[index + n*xsh + o] == segment) {
8                  tmp = v - (float)(*((char*)(raw)+ position + n*xsh + o));
9                  dev += tmp * tmp;
10                 summe += 1;
11                 }
12             }
13         }
14     float var = dev / summe;
15     if (var < 1.0) {
16         var = 1.0;
17         }
18     return var;
19     }
20
21 __device__ float weight(float v,float *raw,float div1,int position) {
22     float tmp = v - (float)(*((char*)(raw) + position));
23     return exp( - tmp * tmp * div1 );
24     }
25
26 __global__ void Funktion(int segment,float *raw,int *slices,float *a,int
       xsh,int ysh,int zsh,int *indices,int sorw,int nbrw) {
27
28     int flat   = xsh * ysh;
29     int column = blockIdx.x * blockDim.x + threadIdx.x;
30     int row    = blockIdx.y * blockDim.y + threadIdx.y;
31     int slice  = blockIdx.z;
32     int plane  = indices[slice];
33     int index  = slice * flat + row * xsh + column;
34
35     if (index < gridDim.z*flat && plane>0 && row>0 && column>0 && plane<
       zsh-1 && row<ysh-1 && column<xsh-1) {
36
37         if (slices[index]==segment) {
38
39             /* Initialisierung und Allokation */
40             float rand;
41             float W0,W1,W2,W3,W4,W5;
42             int n,o,p;
43
44             /* MRG32k3a */
45             float norm = 2.328306549295728e-10;
46             float m1 = 4294967087.0;
47             float m2 = 4294944443.0;
48             float a12 = 1403580.0;
49             float a13n = 810728.0;
```

```
50              float a21 = 527612.0;
51              float a23n = 1370589.0;
52              long k1;
53              float p1, p2;
54              float s10 = index, s11 = index, s12 = index;
55              float s20 = index, s21 = index, s22 = index;
56
57              int k = plane;
58              int l = row;
59              int m = column;
60              int position = plane*flat + row*xsh + column;
61              int step = 0;
62              int n_rw = 0;
63
64              /* Einlesen der Bilddaten */
65              float val = (float)(*((char*)(raw) + position));
66
67              /* Mittlere quadratische Abweichung */
68              float var = varf(position,index,val,raw,segment,slices,xsh);
69              float div1 = 1 / (2 * var);
70
71              /* Berechnung der Random Walks */
72              while (n_rw < nbrw) {
73
74                  /* Berechnung der Gewichte */
75                  W0 = weight(val,raw,div1,position+flat);
76                  W1 = weight(val,raw,div1,position-flat);
77                  W2 = weight(val,raw,div1,position+xsh);
78                  W3 = weight(val,raw,div1,position-xsh);
79                  W4 = weight(val,raw,div1,position+1);
80                  W5 = weight(val,raw,div1,position-1);
81
82                  W1 += W0;
83                  W2 += W1;
84                  W3 += W2;
85                  W4 += W3;
86                  W5 += W4;
87
88                  /* Bestimmung der Zufallszahl mit MRG32k3a */
89
90                  /* Komponent 1 */
91                  p1 = a12 * s11 - a13n * s10;
92                  k1 = p1 / m1;
93                  p1 -= k1 * m1;
94                  if (p1 < 0.0){
95                      p1 += m1;}
96                  s10 = s11;
97                  s11 = s12;
98                  s12 = p1;
99
100                 /* Komponent 2 */
101                 p2 = a21 * s22 - a23n * s20;
102                 k1 = p2 / m2;
103                 p2 -= k1 * m2;
```

```
104                 if (p2 < 0.0){
105                     p2 += m2;}
106                 s20 = s21;
107                 s21 = s22;
108                 s22 = p2;
109
110                 /* Kombination */
111                 if (p1 <= p2) {
112                     rand = W5 * ((p1 - p2 + m1) * norm);
113                     }
114                 else {
115                     rand = W5 * ((p1 - p2) * norm);
116                     }
117
118                 /* Bestimmung der Laufrichtung */
119                 if (rand<W0 || rand==0){n=1; o=0; p=0;}
120                 else if (rand>=W0 && rand<W1){n=-1; o=0; p=0;}
121                 else if (rand>=W1 && rand<W2){n=0; o=1; p=0;}
122                 else if (rand>=W2 && rand<W3){n=0; o=-1; p=0;}
123                 else if (rand>=W3 && rand<W4){n=0; o=0; p=1;}
124                 else if (rand>=W4 && rand<=W5){n=0; o=0; p=-1;}
125
126                 /* Durchfuehrung eines Laufschrittes */
127                 if (k+n>0 && k+n<zsh-1 && l+o>0 &&
128                     l+o<ysh-1 && m+p>0 && m+p<xsh-1) {
129                     k += n; {
130                     l += o;
131                     m += p;
132                     position = k*flat + l*xsh + m; {
133                     atomicAdd(&a[position], 1);
134                     }
135
136                 step += 1;
137
138                 /* Zuruecksetzen des Random Walks */
139                 if (step==sorw) {
140                     k = plane;
141                     l = row;
142                     m = column;
143                     position = k*flat + l*xsh + m;
144                     n_rw += 1;
145                     step = 0;
146                     }
147                 }
148             }
149         }
150     }
```

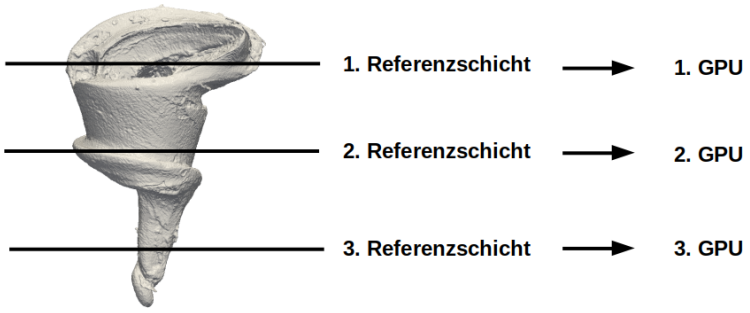

Abbildung 13: Parallele Berechnung verschiedener Referenzschichten. Die Random Walks jeder Schicht werden von jeweils einer Grafikkarte berechnet.

3.2.2 Multi-GPU

Es werden zwei verschiedene Ansätze verwendet, um die GPU-basierten Random Walks unter Verwendung mehrerer Grafikprozessoren (Multi-GPU) zu berechnen. Beim ersten Ansatz werden die vorsegmentierten Schichten auf alle zur Verfügung stehenden GPUs möglichst gleichmäßig aufgeteilt (Abb. 13). Dabei müssen für jedes Voxel alle Treffer, die von den verschiedenen Label ausgehen, gespeichert werden. Für jede GPU wird darum ein Array der Größe $n \times xsh \times ysh \times zsh$ benötigt, wobei n der Anzahl der Label, einschließlich des Hintergrund-Labels, entspricht. Nachdem alle Random Walks berechnet wurden, werden für jedes Voxel alle aus demselben Segment stammenden und von den verschiedenen GPUs berechneten Treffer aufsummiert (Abb. 15). Anschließend wird jedes Voxel dem Segment zugewiesen, von dem die insgesamt höchste Trefferzahl ausging (Abb. 16). Diese Form der Berechnung ist jedoch nur für eine kleine Anzahl von Label und eine geringe Bildgröße möglich, da der benötigte Speicher zum Abspeichern der Treffer $(m \times n \times xsh \times ysh \times zsh \times 32)$-Bit beträgt, wobei m der Anzahl der GPUs entspricht. Dadurch wird die Größe des zur Verfügung stehenden Arbeitsspeichers (RAM) sehr schnell überschritten. Für die Segmentierung der fossilen Wespe (Abschnitt 3.5.7) mit 4 GPUs würde bei diesem Ansatz allein das Speichern der Treffer 2,4 TB benötigen.
Der zweite Ansatz besteht darin, die Bilddaten gleichmäßig in so viele Blöcke zu zerlegen, wie GPUs zur Berechnung vorhanden sind, so dass jede GPU nur die Random Walks berechnet, die in den vorsegmentierten Schichten des ihr zugewiesenen Blocks starten (Abb. 14). Die Random Walks sind dabei jedoch nicht auf das Volumen des Blocks beschränkt, sondern können auch dessen Grenzen überschreiten. Hierfür werden sogenannte *Ghost Blocks*, die sich mit den benachbarten Blöcken überlappen, an das jeweils obere und untere Ende des Blocks angehängt. In den *Ghost Blocks* werden keine Random Walks gestartet. Jedoch werden hier die Treffer der Random Walks bestimmt und an die Prozesse der benachbarten Blöcke übermittelt, um sie dort zu den berechneten Treffern hinzuzufügen.

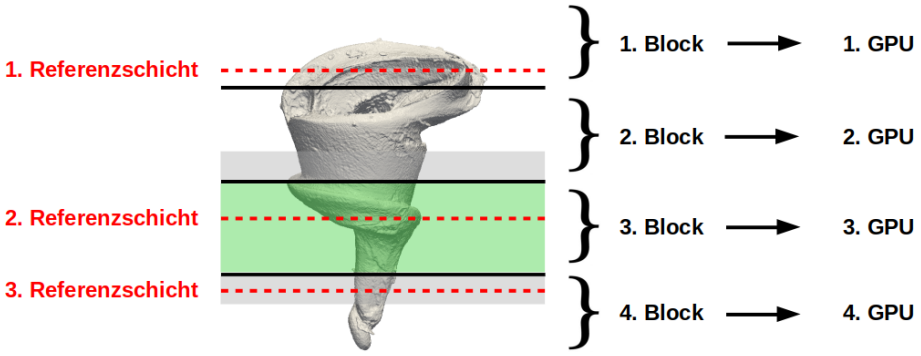

Abbildung 14: Parallele Berechnung mehrerer Blöcke nach einer vorhergehenden Gebietszerlegung. Jede GPU berechnet alle Referenzschichten die sich in dem ihr zugewiesenen Block befinden (grüner Block). Jeder Block wird um *Ghost Blocks* (graue Blöcke) erweitert, die sich mit den benachbarten Blöcken überlappen.

3.3 Strukturerhaltende Glättung

Um das Segmentierungsergebnis zu glätten und gleichzeitig feine Strukturen wie Insektenhaare zu bewahren, wird hier eine Methode entwickelt, die auf der Bewegung von Niveauflächen durch deren mittlere Krümmung beruht [Evans and Spruck, 1991, Osher and Sethian, 1988]. Während übliche Glättungstechniken die morphologischen Operatoren Dilatation und Erosion verwenden, berücksichtigt diese Methode die Anzahl der durch die Random Walks verursachten Treffer eines Pixels und somit implizit die Bilddaten. Dadurch wirkt dieser Ansatz dem Verschwinden feiner Strukturen entgegen, deren Regionen häufig durch Random Walks getroffen wurden und die durch eine morphologische Glättung leicht erodiert werden würden. Die sich aus den Treffern ergebenden Niveauflächen werden dabei jeweils nach dem Berechnen der Random Walks entlang ihrer mittleren Krümmung entwickelt. In diesem Abschnitt soll darum der für diese strukturerhaltende Glättung verwendete mittlere Krümmungsfluss hergeleitet werden. Hierfür wird zunächst die mittlere Krümmung einer Fläche über die Krümmung speziell gewählter Kurven, die die Fläche lokal beschreiben, hergeleitet. Die Herleitung erfolgt dabei angelehnt an [Bär, 2010]. In Satz 3.22 wird dann die Darstellung der mittleren Krümmung als Divergenz des Einheitsnormalenfeldes bewiesen und anschließend daraus der mittlere Krümmungsfluss in Bemerkung 3.23 abgeleitet.

Definition 3.3 (Reguläre Fläche)**.** *Sei $S \subset \mathbb{R}^3$ eine Teilmenge. Die Menge S heißt reguläre Fläche, wenn es zu jedem Punkt $p \in S$ eine offene Umgebung V von p im $\mathbb{R}^3$ gibt, sowie eine offene Teilmenge $U \subset \mathbb{R}^2$ und eine glatte Abbildung $F\colon U \to \mathbb{R}^3$, so dass folgende Bedingungen erfüllt sind:*

1. *$F(U) = S \cap V$ und F ist ein Homöomorphismus.*

2. *Die Jacobi-Matrix D_uF hat für jeden Punkt $u \in U$ Rang 2.*

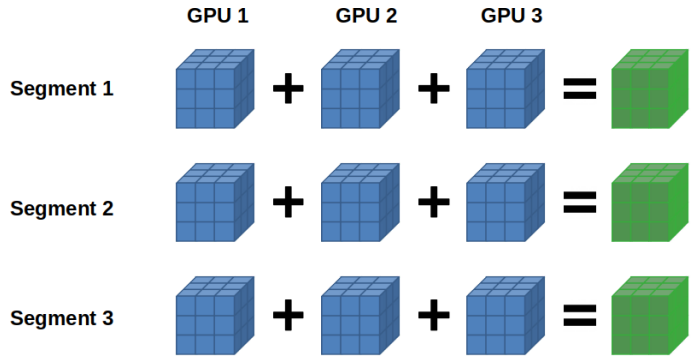

Abbildung 15: Summierung der von Random Walks verursachten Treffer. Für jedes Voxel werden alle Treffer, die von verschiedenen GPUs berechnet wurden und aus demselben Segment stammen, aufsummiert.

Definition 3.4 (Lokale Parametrisierung)**.** *Die Abbildung $F\colon U \to S \cap V$ bzw. das Tripel (U, F, V) heißt lokale Parametrisierung von S um p. Die Menge $S \cap V$ heißt Koordinatenumgebung von p. Die Komponenten u_1 und u_2 von $u = (u_1, u_2)^T$ heißen Koordinaten des Punktes $F(u) \in S$ bzgl. der Parametrisierung F.*

Beispiel 3.5. Sei $U \subset \mathbb{R}^2$ offen, $f\colon U \to \mathbb{R}$ eine glatte Funktion. Die betrachtete Fläche

$$S = \{(x, y, z)^T \in \mathbb{R}^3 \mid (x, y)^T \in U, z = f(x, y)\}$$

ist gegeben durch den Graphen der Funktion f. Sei $V := \mathbb{R}^3$ und $F\colon U \to \mathbb{R}^3$ mit $F(x, y) := (x, y, f(x, y))^T$. Dann gilt $F(U) = S = S \cap V$. Des Weiteren ist F glatt und die Umkehrabbildung $G\colon S \to U$ mit $G(x, y, z) = (x, y)^T$ stetig. Insbesondere ist $F\colon U \to S$ ein Homöomorphismus. Außerdem hat die Jacobi-Matrix

$$D_{(x,y)}F = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ \frac{\partial f}{\partial x}(x, y) & \frac{\partial f}{\partial y}(x, y) \end{pmatrix}$$

für jedes $(x, y)^T \in U$ vollen Rang.

Definition 3.6 (Tangentialebene)**.** *Sei $S \subset \mathbb{R}^3$ eine reguläre Fläche und $p \in S$. Dann heißt*

$$T_pS := \{X \in \mathbb{R}^3 \mid \text{es gibt ein } \varepsilon > 0 \text{ und eine glatte parametrisierte Kurve } c\colon (-\varepsilon, \varepsilon) \to S \text{ mit } c(0) = p \text{ und } \dot{c}(0) = X\}$$

die Tangentialebene von S in p. Die Elemente X der Ebene heißen Tangentialvektoren.

Definition 3.7 (Differential)**.** *Seien S_1, S_2 reguläre Flächen und $f\colon S_1 \to S_2$ eine glatte Abbildung. Das Differential von f in einem Punkt $p \in S_1$ ist die Abbildung*

$$d_pf\colon T_pS_1 \to T_{f(p)}S_2,$$

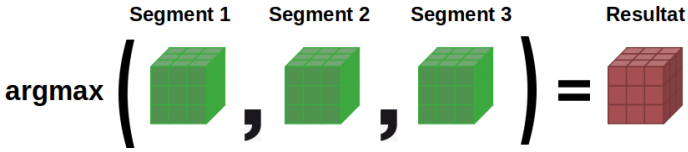

Abbildung 16: Bestimmung des Endergebnisses. Die Segmentierung wird erreicht, indem jedes Voxel dem Segment zugeordnet wird, von dem die höchste Anzahl von Treffern stammt.

mit

$$d_p f(X) := \frac{d}{dt}(f \circ c)\Big|_{t=0} \in T_{f(p)}S_2,$$

wobei $X \in T_pS_1$ *und* $c\colon (-\varepsilon, \varepsilon) \to S_1$ *eine glatte parametrisierte Kurve mit* $c(0) = p$ *und* $\dot{c}(0) = X$.

Bemerkung 3.8. Das Differential ist linear und hängt nicht von der Wahl von c ab.

Definition 3.9 (Orientierbarkeit)**.** *Eine reguläre Fläche* $S \subset \mathbb{R}^3$ *heißt orientierbar, wenn es ein glattes Einheitsnormalenfeld auf* S *gibt.*

Beispiel 3.10. Das Möbiusband hat kein stetiges und damit auch kein glattes Einheitsnormalenfeld.

Definition 3.11 (Weingarten-Abbildung)**.** *Sei* $S \subset \mathbb{R}^3$ *eine orientierbare reguläre Fläche mit Einheitsnormalenfeld* N*, dann heißt der Endomorphismus*

$$W_p\colon T_pS \to T_pS,$$
$$W_p(X) := -d_pN(X),$$

Weingarten-Abbildung.

Definition 3.12 (Erste Fundamentalform)**.** *Sei* $S \subset \mathbb{R}^3$ *eine orientierbare reguläre Fläche, dann heißt die Bilinearform*

$$I_p(X, Y) := g_p(X, Y) = \langle X, Y \rangle,$$

wobei $X, Y \in T_pS$*, die für jeden Punkt* $p \in S$ *das Standardskalarprodukt* $\langle \cdot, \cdot \rangle$ *auf die Tangentialebene* T_pS *einschränkt, d. h.* $g_p := \langle \cdot, \cdot \rangle|_{T_pS \times T_pS}$*, die erste Fundamentalform von* S.

Proposition 3.13. *Sei* $S \subset \mathbb{R}^3$ *eine orientierbare reguläre Fläche mit Weingarten-Abbildung* $W_p\colon T_pS \to T_pS$ *und* $p \in S$*, dann ist* W_p *selbstadjungiert bezüglich der ersten Fundamentalform.*

Beweis. [Bär, 2010, S. 120-122]. □

Definition 3.14 (Zweite Fundamentalform)**.** *Sei $S \subset \mathbb{R}^3$ eine orientierbare reguläre Fläche, W_p die Weingarten-Abbildung und I_p die erste Fundamentalform, dann heißt die Bilinearform*

$$II_p(X,Y) \coloneqq I_p(W_p(X),Y),$$

mit $X,Y \in T_pS$, die zweite Fundamentalform von S in p.

Definition 3.15 (Normalkrümmung)**.** *Sei $S \subset \mathbb{R}^3$ eine orientierbare reguläre Fläche, mit glattem Einheitsnormalenfeld N, p ein Punkt in S und $c\colon (-\varepsilon,\varepsilon) \to S$ eine nach der Bogenlänge parametrisierte Kurve mit $c(0) = p$ und Krümmung $\kappa(0)$ an der Stelle 0. Für $\kappa(0) \neq 0$ ist die Krümmung gegeben durch*

$$\ddot{c}(0) = \kappa(0) \cdot n(0),$$

wobei n der Hauptnormalenvektor an c in 0 ist. Die Normalkrümmung von S in p in Richtung $\dot{c}(0)$ sei definiert als

$$\kappa_{nor} \coloneqq \langle \ddot{c}(0), N(p)\rangle = \begin{cases} \kappa(0) \cdot \langle n(0), N(p)\rangle, & \text{falls } \kappa(0) \neq 0, \\ 0, & \text{falls } \kappa(0) = 0. \end{cases}$$

Satz 3.16 (Satz von Meusnier)**.** *Gegeben seien die Voraussetzungen wie in Definition 3.15. Dann gilt:*

$$\kappa_{nor} = II_p(\dot{c}(0),\dot{c}(0)).$$

Beweis. Da c in S verläuft, gilt

$$\langle N(c(t)), \dot{c}(t)\rangle = 0.$$

Differentiation nach t liefert

$$\begin{aligned} 0 &= \frac{d}{dt}\,\langle N(c(t)),\dot{c}(t)\rangle|_{t=0} \\ &= \left\langle \frac{d}{dt} N(c(t))\Big|_{t=0}, \dot{c}(0) \right\rangle + \langle N(p), \ddot{c}(0)\rangle \\ &\overset{3.7}{=} \langle d_pN(\dot{c}(0)),\dot{c}(0)\rangle + \kappa_{nor} \\ &\overset{3.11}{=} \langle -W_p(\dot{c}(0)),\dot{c}(0)\rangle + \kappa_{nor} \\ &\overset{3.12\ \&\ 3.14}{=} -II_p(\dot{c}(0),\dot{c}(0)) + \kappa_{nor}. \end{aligned}$$

□

Die Normalkrümmung hängt also nur von dem Tangentialvektor von c in 0 ab, nicht aber von der Wahl von c selbst. Alle Kurven in S durch p mit demselben Tangentialvektor haben demnach dieselbe Normalkrümmung.

Bemerkung 3.17. Sei $S \subset \mathbb{R}^3$ eine orientierbare reguläre Fläche, N das Einheitsnormalenfeld, $X \in T_pS$ ein normierter Tangentialvektor an S in p. Der Schnitt von S mit der durch $N(p)$ und X aufgespannten Ebene kann mithilfe des Satzes von der impliziten Funktion durch eine reguläre Kurve c lokal parametrisiert werden. Da in diesem Fall $n(0) = \pm N(p)$, folgt zusammen mit Satz 3.16

$$II_p(X, X) = \pm\kappa(0),$$

wobei κ die Krümmung der ebenen Kurve c ist.

Nach Proposition 3.13 ist W_p selbstadjungiert. Laut dem Spektralsatz existiert somit eine Orthonormalbasis $\{X_1, X_2\}$ von T_pS bestehend aus normierten Eigenvektoren von W_p mit Eigenwerten $\kappa_1, \kappa_2 \in \mathbb{R}$ und es gilt:

$$W_p(X_i) = \kappa_i \cdot X_i, \qquad \text{für } i = 1, 2.$$

Definition 3.18 (Hauptkrümmungen)**.** *Die Eigenwerte κ_1 und κ_2 von W_p heißen Hauptkrümmungen von S in p und die zugehörigen Eigenvektoren $\pm X_i$ für $i = 1, 2$ heißen Hauptkrümmungsrichtungen.*

Bemerkung 3.19. Sei $\kappa_1 \leq \kappa_2, \theta \in \mathbb{R}$ und $X \in T_pS$ ein Einheitsvektor, dann gilt:

$$X = \cos(\theta) \cdot X_1 + \sin(\theta) \cdot X_2,$$

sowie durch einsetzen in II_p:

$$II_p(X, X) = \cos^2(\theta) \cdot \kappa_1 + \sin^2(\theta) \cdot \kappa_2.$$

Nach Satz 3.16 nehmen κ_1 und κ_2 somit jeweils Minimum und Maximum aller Normalkrümmungswerte von S in p an.

Definition 3.20 (Mittlere Krümmung)**.** *Sei $S \subset \mathbb{R}^3$ eine orientierbare reguläre Fläche sowie κ_1 und κ_2 die Hauptkrümmungen von S in p, dann heißt*

$$H(p) := \frac{\kappa_1 + \kappa_2}{2}$$

die mittlere Krümmung von S in p.

Proposition 3.21. *Sei A die Matrixdarstellung der Weingarten-Abbildung W_p, dann gilt für die mittlere Krümmung von S in p:*

$$H = \frac{1}{2}\text{Spur}(A).$$

Beweis. Da nach Proposition 3.13 W_p selbstadjungiert bzgl. I_p ist, folgt aus dem Spektralsatz, dass es eine unitäre Matrix U gibt, so dass $D = U^*AU$, wobei $D = \text{diag}(\kappa_1, \kappa_2)$ eine

Diagonalmatrix mit den Eigenwerten κ_1, κ_2 von A auf der Hauptdiagonalen ist. Da zwei zueinander ähnliche Matrizen die gleichen Eigenwerte und die gleiche Spur besitzen, folgt die Aussage unmittelbar aus der Definition 3.20. □

Satz 3.22. *Sei $\nu = \nabla\Phi/|\nabla\Phi|$ das Einheitsnormalenfeld der durch die Gleichung*

$$\Phi(x, y, f(x, y)) = z - f(x, y) = 0$$

implizit gegebenen Fläche $f(x, y) = z$. Dann gilt für die mittlere Krümmung:

$$H = -\frac{1}{2}\mathrm{div}(\nu).$$

Beweis. Sei $(U \subset \mathbb{R}^2, F, V \coloneqq \mathbb{R}^3)$ eine lokale Parametrisierung der regulären Fläche

$$S = \{(x, y, z)^T \in \mathbb{R}^3 \mid (x, y)^T \in U, z = f(x, y)\}$$

um einen Punkt $p \in S$, wobei $F\colon U \to \mathbb{R}^3$ mit $F(u_1, u_2) = (u_1, u_2, f(u_1, u_2))^T$ (vgl. Beispiel 3.5). Dabei ist die betrachtete Fläche der Graph der Funktion f über dem Parameterbereich U. Ist $\{e_1, e_2\}$ die Standardbasis des $\mathbb{R}^2$, dann bilden $D_uF(u)e_1 = \frac{\partial F}{\partial u_1}(u)$ und $D_uF(u)e_2 = \frac{\partial F}{\partial u_2}(u)$ eine Basis von T_pS, wobei D_uF die Jacobi-Matrix von F beschreibt. Bezüglich dieser Basis ist die Matrixdarstellung der ersten Fundamentalform $I_p(X, Y) \coloneqq g_p(X, Y) = \langle X, Y\rangle$ für $X, Y \in T_pS$ gegeben durch

$$\begin{aligned} g_{ij} &\coloneqq g_p(D_uF(u)e_i, D_uF(u)e_j) \\ &= \left\langle \frac{\partial F}{\partial u_i}(u), \frac{\partial F}{\partial u_j}(u) \right\rangle \end{aligned}$$

und die Matrixdarstellung der zweiten Fundamentalform durch

$$\begin{aligned} h_{ij} &\coloneqq II_p(D_uF(u)e_i, D_uF(u)e_j) \\ &\overset{3.14}{=} I_p(W_p((D_uF(u)e_i), D_uF(u)e_j) \\ &\overset{\text{[Bär, 2010, S. 121]}}{=} \left\langle \frac{\partial^2 F}{\partial u_j \partial u_i}(u), N(p) \right\rangle, \quad i, j = 1, 2, \end{aligned}$$

wobei $N(p) = (D_uF(u)e_1 \times D_uF(u)e_2)/|D_uF(u)e_1 \times D_uF(u)e_2|$ das Einheitsnormalenfeld der lokalen Parametrisierung $F\colon U \subset \mathbb{R}^2 \to \mathbb{R}^3$ ist. Die Matrixdarstellung der Weingarten-Abbildung W_p lautet folglich:

$$\begin{aligned} A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} &= \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix}^{-1} \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix} \\ &= \frac{1}{g_{11}g_{22} - g_{12}^2} \begin{pmatrix} g_{22} & -g_{12} \\ -g_{21} & g_{11} \end{pmatrix} \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix} \end{aligned} \tag{3.3}$$

und für die Matrixelemente gilt:

$$g_{11} = 1 + f_{u_1}^2 \tag{3.4}$$
$$g_{12} = g_{21} = f_{u_1} f_{u_2} \tag{3.5}$$
$$g_{22} = 1 + f_{u_2}^2 \tag{3.6}$$
$$h_{11} = f_{u_1 u_1} / \sqrt{1 + f_{u_1}^2 + f_{u_2}^2} \tag{3.7}$$
$$h_{12} = h_{21} = f_{u_1 u_2} / \sqrt{1 + f_{u_1}^2 + f_{u_2}^2} \tag{3.8}$$
$$h_{22} = f_{u_2 u_2} / \sqrt{1 + f_{u_1}^2 + f_{u_2}^2}. \tag{3.9}$$

Sei nun $\Phi(x, y, f(x, y)) = \alpha$ die implizite Darstellung der Fläche $f(x, y) = z$ zum Niveau α. Durch Differentiation von Φ nach x und y lassen sich die Ableitungen $f_x, f_y, f_{xx}, f_{xy}, f_{yy}$ durch die Ableitungen von Φ wie folgt darstellen:

$$\begin{aligned}
f_x &= -\Phi_x/\Phi_z \\
f_y &= -\Phi_y/\Phi_z \\
f_{xx} &= -(\Phi_{xx} + \Phi_{xz} f_x)\Phi_z^{-1} + \Phi_x(\Phi_z)^{-2}(\Phi_{zx} + \Phi_{zz} f_x) \\
&= -(\Phi_{xx}/\Phi_z - \Phi_{xz}\Phi_x/\Phi_z^2) + \Phi_x(\Phi_{zx}/\Phi_z^2 - \Phi_{zz}\Phi_x/\Phi_z^3) \\
&= -(\Phi_z^2\Phi_{xx} - 2\Phi_x\Phi_z\Phi_{xz} + \Phi_x^2\Phi_{zz})/\Phi_z^3 \\
f_{yy} &= -(\Phi_z^2\Phi_{yy} - 2\Phi_y\Phi_z\Phi_{yz} + \Phi_y^2\Phi_{zz})/\Phi_z^3 \\
f_{xy} &= -(\Phi_{xy} + \Phi_{xz} f_y)\Phi_z^{-1} + \Phi_x(\Phi_z)^{-2}(\Phi_{zy} + \Phi_{zz} f_y) \\
&= -(\Phi_z^2\Phi_{xy} - \Phi_y\Phi_z\Phi_{xz} - \Phi_x\Phi_z\Phi_{yz} + \Phi_x\Phi_y\Phi_{zz})/\Phi_z^3.
\end{aligned}$$

Insbesondere sind diese Darstellungen unabhängig von α. Sei $|\nabla\Phi| \neq 0$ und insbesondere auch $\Phi_z \neq 0$. Proposition 3.21 liefert dann zusammen mit den Gleichungen 3.3-3.9 und den Darstellungen von $f_x, f_y, f_{xx}, f_{xy}, f_{yy}$ die Behauptung:

$$\begin{aligned}
H &\overset{3.21}{=} \frac{1}{2}\mathrm{Spur}(A) \\
&\overset{3.3}{=} \frac{1}{2}\frac{h_{11}g_{22} - 2h_{12}g_{12} + h_{22}g_{11}}{g_{11}g_{22} - g_{12}^2} \\
&= \frac{1}{2}\frac{f_{xx}(1 + f_y^2) - 2f_{xy}f_x f_y + f_{yy}(1 + f_x^2)}{(1 + f_x^2 + f_y^2)^{3/2}} \\
&= -\frac{1}{2}\left(\frac{\Phi_{xx}(\Phi_y^2 + \Phi_z^2) + \Phi_{yy}(\Phi_x^2 + \Phi_z^2) + \Phi_{zz}(\Phi_x^2 + \Phi_y^2)}{(\Phi_x^2 + \Phi_y^2 + \Phi_z^2)^{3/2}}\right. \\
&\qquad \left. + \frac{-2\Phi_{xy}\Phi_x\Phi_y - 2\Phi_{xz}\Phi_x\Phi_z - 2\Phi_{yz}\Phi_y\Phi_z}{(\Phi_x^2 + \Phi_y^2 + \Phi_z^2)^{3/2}}\right) \\
&= -\frac{1}{2}\mathrm{div}\left(\frac{\nabla\Phi}{|\nabla\Phi|}\right) = -\frac{1}{2}\mathrm{div}(\nu).
\end{aligned}$$

□

Bemerkung 3.23. Für die Verformung der Niveauflächen sei Φ eine von der Zeit $t \in [0, T)$ abhängige Funktion $\Phi\colon \mathbb{R}^3 \times [0, T) \to \mathbb{R}$, wobei $\Phi(x, 0)$ der Anzahl der Treffer des Voxels an der Stelle $x \in \mathbb{R}^3$ entspricht. Für festes $\alpha \in \mathbb{R}$ sei

$$\Gamma_\alpha(t) \coloneqq \{x \in U \subset \mathbb{R}^3 \mid \Phi(x, t) = \alpha\}$$

die Niveaufläche zum Zeitpunkt t und Niveau α. Sei nun

$$\nu(x, t) = \frac{\nabla\Phi(x, t)}{|\nabla\Phi(x, t)|}$$

ein glattes Einheitsnormalenfeld von $\Gamma_\alpha(t)$ in U, dann folgt aus Satz 3.22:

$$F = -\frac{1}{2}\mathrm{div}(\nu)\nu$$

ist das mittlere Krümmungsvektorfeld von $\Gamma_\alpha(t)$. Für ein festes $t \geq 0$ entwickelt sich ein Punkt $x \in \Gamma_\alpha(t) \cap U$ entlang der Lösungskurven der durch das Vektorfeld F gegebenen gewöhnlichen Differentialgleichung

$$\begin{cases} \dot{x}(s) = -[\mathrm{div}(\nu)\nu](x(s), s) & \text{für } s > t, \\ x(t) = x. \end{cases}$$

Da $x(s) \in \Gamma_\alpha(s)$ für $s \geq t$, gilt $\Phi(x(s), s) = \alpha$. Somit folgt

$$0 = \frac{d}{ds}\Phi(x(s), s) = -[(\nabla\Phi \cdot \nu)\mathrm{div}(\nu)](x(s), s) + \Phi_t(x(s), s).$$

Für $s = t$ folgt für alle (x, t)

$$\Phi_t = (\nabla\Phi \cdot \nu)\mathrm{div}(\nu).$$

Durch Einsetzen von ν ergibt sich dadurch die Entwicklung von Φ entlang der mittleren Krümmung in Form der partiellen Differentialgleichung

$$\frac{\partial\Phi}{\partial t} = |\nabla\Phi|\,\mathrm{div}\left(\frac{\nabla\Phi}{|\nabla\Phi|}\right). \tag{3.10}$$

Zur Vermeidung von Datentransfer wird die Differentialgleichung 3.10 direkt nach der Berechnung der Random Walks auf die sich im Speicher der GPU befindlichen Treffer Φ angewandt und mithilfe des expliziten Euler-Verfahrens gelöst. Die Gleichung lässt sich mithilfe der Differenzenquotienten 2.7-2.9 analog zur Diskretisierung 2.11 der partiellen Differentialgleichung 2.3 approximieren. Durch einen auf der rechten Seite hinzugefügten skalaren Wert μ sowie über die Anzahl der durchgeführten Zeitschritte, kann die Stärke der strukturerhaltenden Glättung beeinflusst werden. Die nachfolgenden Ergebnisse basieren alle auf der Segmentierung ohne strukturerhaltende Glättung. Das geglättete Ergebnis wird jedoch von der Online-Plattform Biomedisa (Kapitel 5) für den jeweiligen Fall optional bereitgestellt. Hierfür ist $\mu = 1$, $\Delta t = 0.01$ und $T = 1$, d. h. es werden 100 Zeitschritte mit Schrittweite $\Delta t = 0.01$ durchgeführt.

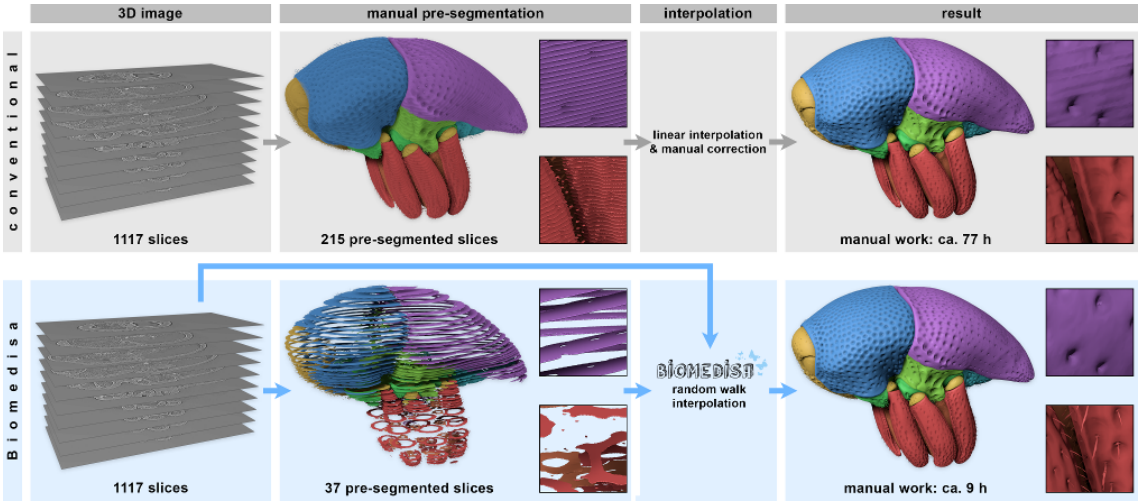

Abbildung 17: Vergleich eines traditionellen Segmentierungsverfahrens (oben) und der Segmentierung durch die GPU-basierten Random Walks (unten). Mit der häufig verwendeten linearen Interpolation wurden insgesamt 77 Stunden für die Rekonstruktion des Rüsselkäfers benötigt. Die GPU-basierten Random Walks erforderten lediglich 9 Stunden manuelle Arbeitszeit. Beide Verfahren benötigen eine manuelle Vorsegmentierung. Die GPU-basierten Random Walks berücksichtigen im Gegensatz zur linearen Interpolation auch die Bilddaten und ermöglichen somit einen wesentlich geringeren manuellen Aufwand. Darüber hinaus können Interpolationsartefakte vermieden und feine Strukturen, wie Haare, rekonstruiert werden, https://www.youtube.com/watch?v=2Vc0zJxv42g [Lösel et al., 2020].

3.4 Segmentierung eines *Trigonopterus* Rüsselkäfers

Das Vorgehen des Segmentierungsverfahrens soll hier exemplarisch anhand eines *Trigonopterus* Rüsselkäfers dargestellt werden (Abb. 17-19 & 22, https://www.youtube.com/watch?v=uNoyAPkCKnI). Die volumetrischen Bilddaten des Käfers wurden mithilfe von Synchrotron-Röntgen-Mikrotomographie (SR-μCT) am Karlsruher Elektronen-Synchrotron ANKA erzeugt. Zur besseren manuellen Bearbeitung wurde der Käfer innerhalb des Volumens ausgerichtet und der ursprüngliche Bilddatensatz von $2016 \times 2016 \times 2016$ Voxel auf eine Größe von $1497 \times 734 \times 1117$ Voxel zugeschnitten. Um eine dreidimensionale Rekonstruktion, bestehend aus 64 Einzelteilen, des *Trigonopterus* zu erhalten und einen Vergleich mit dem, bei vielen Wissenschaftlern/-innen nach wie vor vorherrschenden, fast ausschließlich manuellen Ansatz zu ermöglichen, wurde, wie hier üblich, zunächst jede 5. Schicht des Datensatzes von einem/einer biologischen Experten/-in manuell vorsegmentiert. Die daraus resultierenden 215 Schichten wurden dann mit Amira 5.6 linear interpoliert und das Ergebnis anschließend aufwendig manuell nachbearbeitet (Abb. 17 oben). Insgesamt dauerte diese Vorgehensweise 77 Stunden, wobei 52 Stunden auf die manuelle Vorsegmentierung und 25 Stunden auf die notwendige Korrektur des Interpolationsergebnisses entfielen. Im Gegensatz dazu genügten 37 dieser vorsegmentierten Schichten, um mit dem Ansatz der GPU-basierten Random Walks eine hoch akkurate Segmentierung zu er-

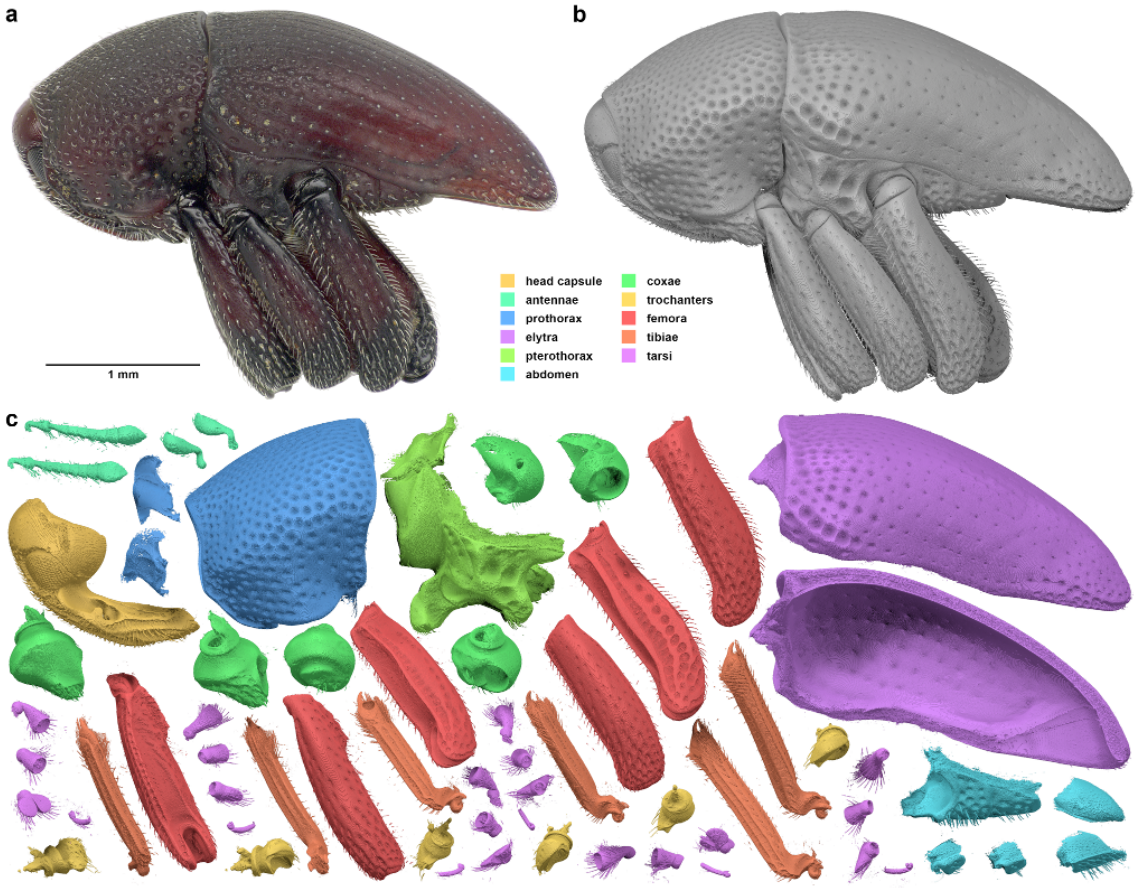

Abbildung 18: Segmentierung eines *Trigonopterus* Rüsselkäfers. a Photo des ursprünglichen Käfers. **b** Ergebnis der Segmentierung basierend auf 37 vorsegmentierten und entsprechend der Morphologie des Käfers gewählten Schichten. **c** Die 64 rekonstruierten Körperteile des Käfers [Lösel et al., 2020].

halten (Abb. 17 unten & 19). Die Anzahl der hierfür benötigten Schichten entsprach einem manuellen Arbeitsaufwand von 9 Stunden. Zudem konnten typische Interpolationsartefakte der linearen Interpolation vermieden und selbst feine Strukturen wie Haare rekonstruiert werden (Abb. 17 & 18). Die Wahl der 37 Schichten erfolgte entsprechend der Morphologie des Käfers. Dabei konnten in den oberen (oberes Pronotum & Elytren) und unteren (distale Teile der Beine) Bereichen größere Abstände zwischen den Schichten gelassen werden als im Bereich des Thorax mit den Schraubengelenken.

3.5 Numerische Ergebnisse & Experimente

Neben der Segmentierung eines *Trigonopterus* Rüsselkäfers (Abschnitt 3.4) wurde das Verfahren erfolgreich zur Segmentierung vieler weiterer Datensätze, resultierend aus μCT, SR-

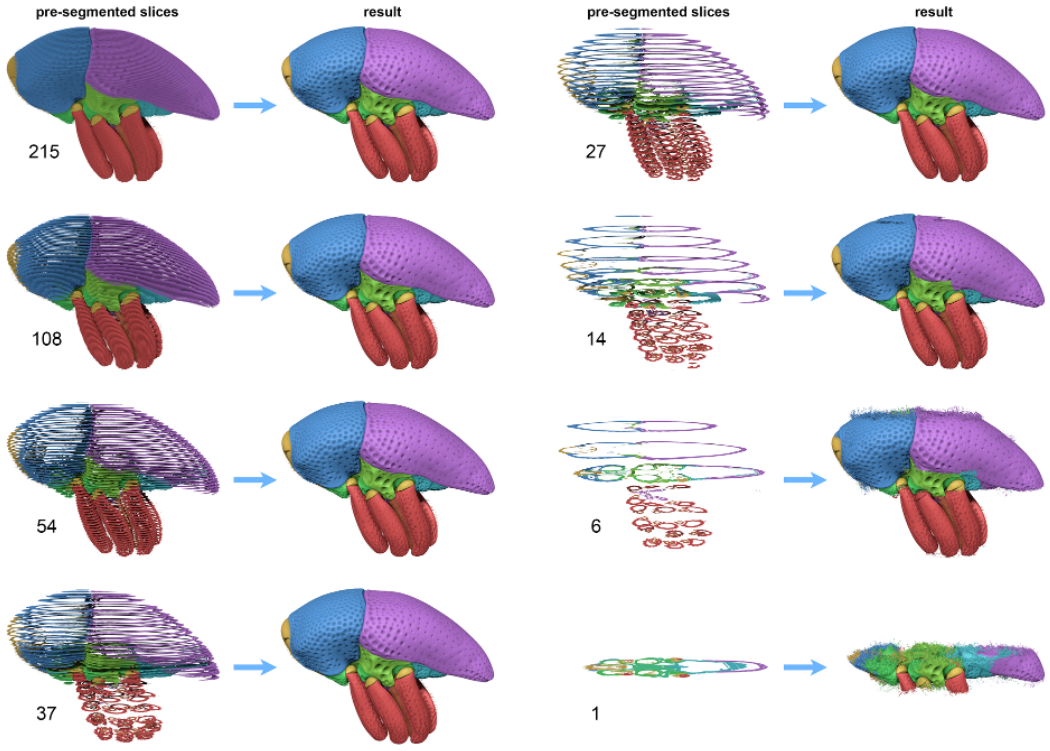

Abbildung 19: Segmentierung eines *Trigonopterus* Rüsselkäfers basierend auf einer unterschiedlichen Anzahl vorsegmentierter Schichten. Die Verwendung von 215, 108 und 54 gleichmäßig verteilten Schichten, die einer Vorsegmentierung jeder 5., 10. und 20. Schicht entsprechen, lieferte genaue Ergebnisse. Durch eine an die Morphologie des Rüsselkäfers angepasste Wahl des Abstands zwischen den Schichten reichten bereits 37 Schichten für ein Ergebnis von gleicher Qualität. Eine geringere Anzahl vorsegmentierter Schichten führte zu zunehmend fehlerbehafteten Resultaten [Lösel et al., 2020].

μCT und MRT, eingesetzt. Darunter eine in Bernstein eingeschlossene fossile Wespe (Abb. 20d), eine ebenfalls in Bernstein eingeschlossene Theropoden Dinosaurierklaue (Abb. 20g), eine australische Bulldoggenameisenkönigin (Abb. 20h), eine Fauchschabe (Abb. 20e), ein Medaka (Japanischer Reisfisch) (Abb. 20a), menschliche Herzen [Lösel and Heuveline, 2017] (Abb. 20c) und Mäusebackenzähne [Balanta-Melo et al., 2019] (Abb. 20b). Darüber hinaus wurde es in einer großen vergleichenden Studie zur Artenbeschreibung fossiler parasitärer Schlupfwespen in mineralisierten Fliegenpuparien eingesetzt [van de Kamp et al., 2018] (Abb. 20f). Alle Datensätze dienen als Grundlage für numerische Experimente (3.5.11 & 3.5.12) und in Abschnitt 3.7 der Evaluation sowie dem Vergleich mit den in Kapitel 2 beschriebenen Methoden. Die tomographischen Rekonstruktionen der SR-μCT-Scans und des Medakas wurden mit einem GPU-beschleunigten gefilterten Rückprojektionsalgorithmus erstellt, der im UFO-Software-Framework implementiert ist [Vogelgesang et al., 2016]. Anschließend wurden die Rekonstruktionen von 32-Bit auf 8-Bit konvertiert, um die Datengröße für eine leichtere Handhabung zu reduzieren. Alle Datensätze wurden ausgerichtet

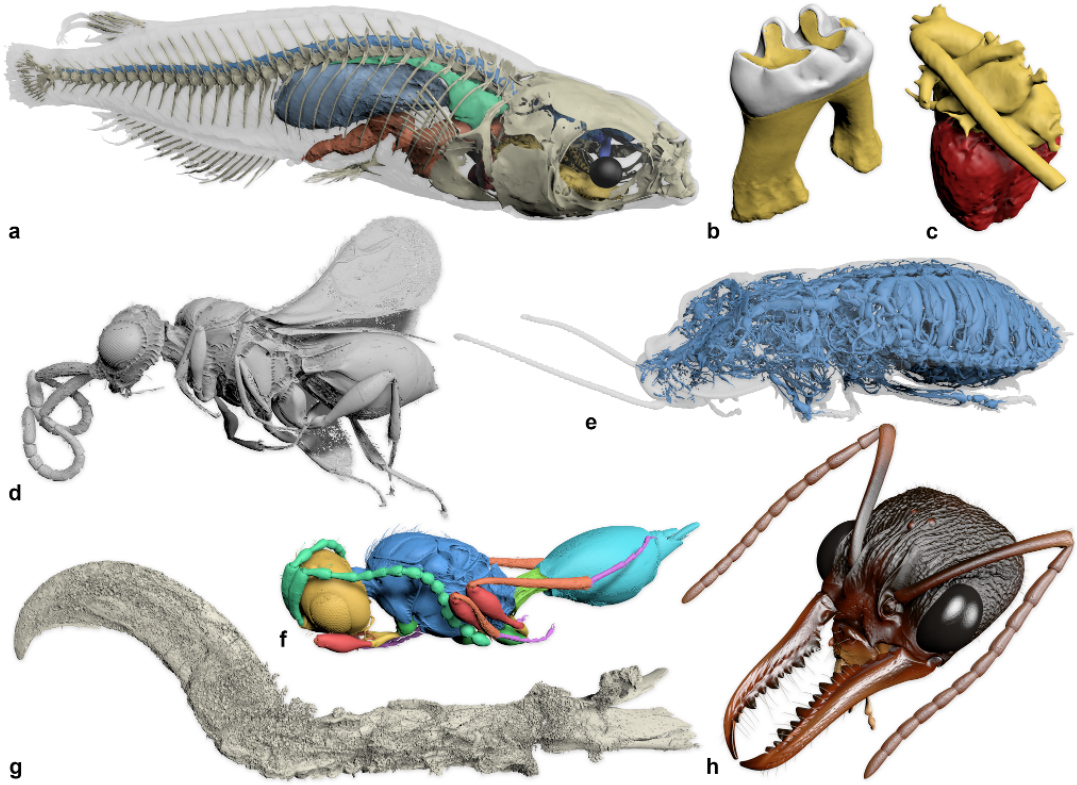

Abbildung 20: Segmentierungen biologischer und medizinischer Datensätze.
a Japanischer Reisfisch (μCT). **b** Zahnschmelz (weiß) und Dentin (gelb) eines Mäusebackenzahns (μCT). **c** Herzmuskel und Blutgefäße eines menschlichen Herzens (MRT). **d** Parasitäre Wespe segmentiert aus baltischem Bernstein (SR-μCT). **e** Tracheensystem einer Fauchschabe (μCT). **f** Theropodenklaue segmentiert aus burmesischem Bernstein (SR-μCT). **g** Parasitäre Wespe eingeschlossen in einer fossilen Fliegenpuppe (SR-μCT). **h** Australische Bulldoggenameisenkönigin (SR-μCT), https://www.youtube.com/watch?v=gKUa78LAMms [Lösel et al., 2020].

und beschnitten, um die Probenorientierung und -position innerhalb des Volumens für die manuelle Vorsegmentierung zu optimieren. Zur Erstellung der Abbildungen wurden Ausreißer entfernt, alle Label mit Amira 5.6 in Polygonnetze umgewandelt, als OBJ-Dateien exportiert und in CINEMA 4D R20 wieder zusammengesetzt. CINEMA 4D R20 wurde zur Glättung, Polygonreduktion und zum Rendern der Abbildungen verwendet.

3.5.1 *Trigonopterus* Rüsselkäfer (Abb. 17-19 & 22)

Ein papuanischer Rüsselkäfer (*Trigonopterus* sp. [Coleoptera: Curculionidae]) aus der Sammlung des Staatlichen Museums für Naturkunde Karlsruhe (Volumengröße: $1497 \times 734 \times 1117$ Voxel) wurde in 100 % Ethanol fixiert und in seiner Abwehrposition an der UFO-Aufnahmestation der KIT-Lichtquelle gescannt (Scanparameter siehe [Lösel et al., 2020]).

Insgesamt wurden 64 Körperteile mit Amira 5.6 in jeder 5. Schicht (= 215 Schichten) manuell vorsegmentiert. Verschiedene Teilmengen dieser vorsegmentierten Schichten dienten als Initialisierung (Abb. 19), um die Effizienz der GPU-basierten Random Walks im Vergleich zur konventionellen morphologischen Interpolation zwischen den vorsegmentierten Schichten zu ermitteln. Die Berechnung für 37 ausgewählte vorsegmentierte Schichten dauerte 19 min 43 s auf 1 NVIDIA Tesla V100 und 10 min 22 s auf 4 NVIDIA Tesla V100.

3.5.2 Medaka (Abb. 20a)

Ein Japanischer Reisfisch (*Oryzias latipes*) mit 0,33 % Phosphorwolframsäure (PTA) und 0,3 % Lugols Jod (I3K) gefärbt und in 4 % Agarose eingebettet (Volumengröße: 900 × 1303 × 4327 Voxel) wurde mit dem Labor-Röntgenaufbau des Instituts für Photonenforschung und Synchrotronstrahlung des KIT gescannt (Scanparameter siehe [Lösel et al., 2020]). Die Tierhaltung und Versuchsdurchführung wurde am Institut für Biologische und Chemische Systeme (IBCS) des Karlsruher Instituts für Technologie (KIT) nach den deutschen Tierschutzbestimmungen durchgeführt (Regierungspräsidium Karlsruhe, Deutschland; Tierschutzgesetz 111, Abs. 1, Nr. 1, AZ35-9185.64/BH). Das IBCS steht unter der Aufsicht des Regierungspräsidiums Karlsruhe, das das Versuchsverfahren genehmigt hat. Es wurden 11 Körperteile in insgesamt 231 Schichten vorsegmentiert. Die verschiedenen Körperteile wurden nicht immer in den gleichen Schichten vorsegmentiert. Daher wurde jedes Label separat berechnet, was zu Rechenzeiten zwischen 4 min 32 s und 57 min 30 s mit 1 NVIDIA Tesla V100 und zwischen 4 min 48 s und 24 min 14 s mit 4 NVIDIA Tesla V100 führte.

3.5.3 Mäusebackenzähne (Abb. 20b)

Zehn Mäuseunterkiefer wurden ex-vivo mit einem diondo d3 μCT-Gerät (Volumengröße im Durchschnitt: 438 × 543 × 418 Voxel) gescannt und analysiert [Balanta-Melo et al., 2019]. Jede 20. Schicht (im Durchschnitt 20 Schichten pro Volumen) wurde mit Avizo 9.2 vorsegmentiert. Zur Differenzierung und Vorsegmentierung von Schmelz, Dentin und Alveolarknochen wurde eine Schwellenwert-basierte Selektion mit manueller Korrektur verwendet. Die Berechnung dauerte durchschnittlich $143,5 \pm 11,3$ s auf 1 NVIDIA Tesla V100 und $53,6 \pm 4,2$ s auf 4 NVIDIA Tesla V100.

3.5.4 Menschliche Herzen (Abb. 20c)

Während der klinischen Praxis im Boston Children's Hospital, Boston, MA, USA, wurden 20 kardiovaskuläre 3D-Magnetresonanz-Bilder (CMR) aufgenommen. Die Fälle umfassen eine Vielzahl von angeborenen Herzfehlern. Die Bildgebung erfolgte in axialer Ansicht auf

einem 1,5T Philips Achieva Scanner ($\mathrm{TR} = 3,4$ ms, $\mathrm{TE} = 1,7$ ms, $\alpha = 60°$) ohne Kontrastmittel unter Verwendung einer Steady-State-Sequenz (*Steady-State Free Precession Pulse Sequence*). Die Daten wurden von den Organisatoren des MICCAI Workshops *Whole-Heart and Great Vessel Segmentation from 3D Cardiovascular MRI in Congenital Heart Disease* (HVSMR 2016, `http://segchd.csail.mit.edu/`) bereitgestellt [Pace et al., 2015]. Zehn Trainings- (einschließlich *Ground-Truth*-Label) und zehn Test-CMR-Scans wurden entweder als vollständiges axiales CMR-Bild, als auf das Herz und die Aorta zugeschnittener Datensatz oder als *Short Axis View* bereitgestellt. Die *Ground-Truth*-Label wurden von einem/einer geschulten Auswerter/-in erstellt und von zwei klinischen Experten/-innen validiert. Die Bildgrößen und Bildauflösungen variieren und betragen durchschnittlich $157 \times 216 \times 167$ Voxel bzw. $0,82 \times 0,82 \times 0,87$ mm^3. Zwei Objekte, das Myokard und die Blutgefäße, wurden segmentiert. In dieser Arbeit wurden ausschließlich die 10 auf das Herz und die Aorta zugeschnittenen Trainingsbilder berücksichtigt. Bei einer Vorsegmentierung jeder 20. Schicht betrug die Rechenzeit durchschnittlich $3,4 \pm 2,4$ s auf 1 NVIDIA Tesla V100 und $2,2 \pm 1,1$ s auf 4 NVIDIA Tesla V100.

3.5.5 Wespe in Bernstein (Abb. 20d)

Eine Ceraphronoidwespe (*Conostigmus talamasi* [Hymenoptera: Ceraphronidae]), eingeschlossen in einem ca. 33-55 Millionen Jahre alten baltischen Bernstein aus dem Eozän (Volumengröße: $1417 \times 2063 \times 2733$ Voxel) [Mikó et al., 2018b]. Das Exemplar wird beim Senckenberg Deutschen Entomologischen Institut (Müncheberg, Deutschland) mit der Sammlungsnummer DEI-GISHym31819 aufbewahrt. Die Probe wurde aufgrund ihrer Größe in drei Schritten (oberer, mittlerer und unterer Teil) an der UFO-Aufnahmestation der KIT-Lichtquelle gescannt (Scanparameter siehe [Lösel et al., 2020]). Die einzelnen Aufnahmen wurden anschließend zu einem einzigen volumetrischen Bild kombiniert. Insgesamt wurden 15 Körperteile in jeder 20. Schicht vorsegmentiert, was zu 126 vorsegmentierten Schichten und einer Rechenzeit von 79 min 17 s auf 1 NVIDIA Tesla V100 und 31 min 48 s auf 4 NVIDIA Tesla V100 führte.

3.5.6 Fauchschabe (Abb. 20e)

Eine Zwergfauchschabe (*Elliptorhina Chocardi* [Blattodea: Blaberidae]) wurde in 100 % Ethanol fixiert und mit dem Labor-Röntgenaufbau des Instituts für Photonenforschung und Synchrotronstrahlung des KIT gescannt (Volumengröße: $613 \times 606 \times 1927$ Voxel, Scanparameter siehe [Lösel et al., 2020]). Ein einzelnes Segment, das das Trachealnetzwerk repräsentiert, wurde in jeder 25. Schicht vorsegmentiert, was zu 55 vorsegmentierten Schichten führte. Die Berechnung dauerte 8 min 23 s auf 1 NVIDIA Tesla V100 und 3 min 17 s auf 4 NVIDIA Tesla V100.

3.5.7 Fossile Wespe (Abb. 20f)

Eine ca. 34-40 Millionen Jahre alte parasitoide Wespe (*Xenomorphia resurrecta* [Hymenoptera: Diapriidae]), enthalten in einer mineralisierten Fliegenpuppe aus dem Paläogen Frankreichs (Volumengröße: $1077 \times 992 \times 2553$ Voxel), wurde in zwei Schritten (oberer und unterer Teil) an der UFO-Bildgebungsstation der KIT-Lichtquelle gescannt (Scanparameter siehe [Lösel et al., 2020]). Das Exemplar wird im Naturhistorischen Museum Basel unter der Sammlungsnummer NMB F2875 aufbewahrt. Insgesamt wurden 56 einzelne Label in jeder 10. Schicht vorsegmentiert, was zu 223 vorsegmentierten Schichten und einer Rechenzeit von 55 min 57 s auf 1 NVIDIA Tesla V100 und 21 min 46 s auf 4 NVIDIA Tesla V100 führte. Die Wespe wurde erstmals in einer im Jahr 2018 veröffentlichten Studie mit ähnlichen Exemplaren, die auf die gleiche Weise verarbeitet wurden, beschrieben [van de Kamp et al., 2018].

3.5.8 Theropodenklaue (Abb. 20g)

Die Klaue eines unbekannten juvenilen Theropoden-Dinosauriers [Theropoda: Coelurosauria], eingeschlossen in burmesischem Bernstein aus der Sammlung von Patrick Müller, Käshofen (Volumengröße: $1986 \times 1986 \times 3602$ Voxel), wurde in zwei Schritten (oberer und unterer Teil) an der UFO-Bildgebungsstation der KIT-Lichtquelle gescannt (Scanparameter siehe [Lösel et al., 2020]). Ein einzelnes Segment, das die Klaue repräsentiert, wurde in jeder 40. Schicht vorsegmentiert, was zu 88 vorsegmentierten Schichten führte. Die Berechnung dauerte 29 min 30 s auf 1 NVIDIA Tesla V100 und 16 min 2 s auf 4 NVIDIA Tesla V100.

3.5.9 Bulldoggenameisenkönigin (Abb. 20h)

Der Kopf einer australischen Bulldoggenameisenkönigin (*Myrmecia pyriformis* [Hymenoptera: Formicidae]) wurde an der UFO-Bildgebungsstation der KIT-Lichtquelle gescannt (Scanparameter siehe [Lösel et al., 2020], Volumengröße: $1957 \times 1165 \times 2321$ Voxel). Zu beachten ist, dass die lange Belichtungszeit für diese Art von Präparat nicht optimal war und dies zu Blasenbildung im Weichgewebe, nachträglichen Artefakten im Tomogramm und allgemein verrauschten Daten führte. Artefakte durch diffuse Kanten im Tomogramm erforderten leichte manuelle Korrekturen. Insgesamt wurden 52 Körperteile vorsegmentiert. Der Kopf, die linke Antenne und die rechte Antenne wurden getrennt segmentiert. Der Abstand zwischen den vorsegmentierten Schichten wurde je nach Notwendigkeit zwischen 5 und 50 Schichten gewählt. Für den Kopf wurden insgesamt 81 Schichten, für die linke Antenne 109 Schichten und für die rechte Antenne 114 Schichten vorsegmentiert. Die Berechnungen dauerten 42 min 15 s, 11 min 10 s und 16 min 52 s auf 1 NVIDIA Tesla V100 bzw. 14 min 54 s, 5 min 24 s und 6 min 41 s auf 4 NVIDIA Tesla V100. Das aus

der Segmentierung resultierende Oberflächennetz wurde mit CINEMA 4D R20 künstlich eingefärbt und animiert.

3.5.10 Menschliche Kieferknochen (Abb. 25)

Mehrere Computertomographie-Datensätze des kranio-maxillo-fazialen Komplexes wurden im klinischen Alltag der Klinik für Mund-, Kiefer- und Gesichtschirurgie der Medizinischen Universität Graz in Österreich erhoben [Wallner et al., 2019]. Zehn dieser CT-Scans wurden ausgewählt und die jeweiligen Unterkiefer wurden von zwei klinischen Experten/-innen mit MeVisLab vollständig manuell segmentiert [Wallner et al., 2018]. Die Segmentierungen wurden als Kontursegmentierungsobjekte (CSO-Dateien) gespeichert. Um die Daten verarbeiten zu können, wurden die CSO-Dateien mit MeVisLab in das NRRD-Dateiformat konvertiert. Anschließend wurden die Daten mit *CSOLoad* und die entsprechenden CT-Daten mit *itkImageFileReader* geladen. Beide dienten als Eingabe für die Funktion *CSOConvertToImage*. Schließlich konnte die so konvertierte CSO-Datei mit *itkImageFileWriter* als NRRD abgespeichert werden. Die CT-Scans wurden mit einem medizinischen Somatom Sensation 64 Scanner von Siemens durchgeführt (im Durchschnitt: Scan-Dosis = 120 kV, Scan-Exposition = 285,5 mAs). Bildgröße und Bildauflösung betrugen durchschnittlich $512 \times 512 \times 195$ Voxel bzw. $0,453 \times 0,453 \times 1,3$ mm^3. Unter Verwendung von ungefähr jeder 20. Schicht der vollständig manuell segmentierten Daten zur Initialisierung (was jeweils 5-8 vorsegmentierten Schichten entspricht) betrug die durchschnittliche Rechenzeit $79 \pm 15,5$ s auf 1 NVIDIA Tesla V100 und $31,6 \pm 3,6$ s auf 4 NVIDIA Tesla V100.

3.5.11 Adaptive Berechnung

Um die Rechenzeit deutlich zu reduzieren, ohne dabei die Qualität des Ergebnisses zu beeinträchtigen, kann die Anzahl der Random Walks, die in einem Pixel starten, basierend auf der Vorsegmentierung angepasst werden. Für eine qualitativ hochwertige Segmentierung sind insbesondere die Random Walks in der Nähe der Objektkanten von entscheidender Bedeutung, da diese in direkter Konkurrenz zu den Random Walks der benachbarten Label stehen. Wohingegen die Random Walks, die weit im Inneren des Objekts beginnen, kaum mit anderen Random Walks konkurrieren. Daher wird hier eine reduzierte Anzahl an Random Walks verwendet, um die Gesamtberechnungszeit zu verkürzen. Eine Startposition wird dabei als im inneren Bereich liegend betrachtet, wenn sich in einem Bereich von 101×101 Pixel um die Startposition herum kein anderes Label befindet. Bei gleichwertiger Genauigkeit waren die Segmentierungen der Beispieldatensätze *Trigonopterus* Rüsselkäfer 3.5.1, Wespe aus Bernstein 3.5.5, Fauchschabe 3.5.6, Theropodenklaue 3.5.8, mineralisierte Wespe 3.5.7 und Bulldoggenameisenkönigin 3.5.9 mit einer auf diese Weise vorgenommenen Anpassung im Durchschnitt 45 % schneller.

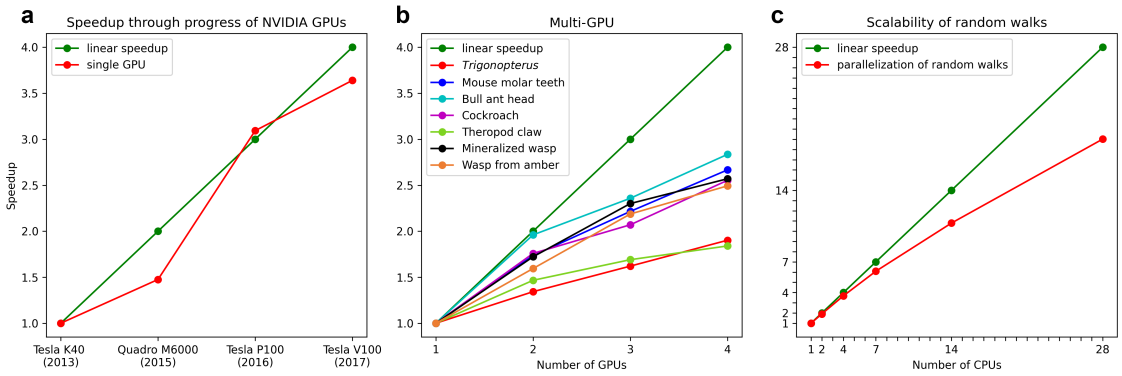

Abbildung 21: Unterschiedliche Speedup-Szenarien der GPU-basierten Random Walk Berechnung. a Aufgrund einer fortschreitenden technologischen Entwicklung. **b** Durch Erhöhung der Anzahl an Grafikkarten. **c** Durch Erhöhung der Anzahl einzelner Recheneinheiten (hier: zentrale Recheneinheiten) [Lösel et al., 2020].

3.5.12 Skalierbarkeit

Da es nicht möglich ist, einzelne Recheneinheiten (*Compute Units*) einer GPU zu deaktivieren, wurde hier repräsentativ PyOpenCL [Klöckner et al., 2012] zusammen mit 28 CPUs verwendet, um die Skalierbarkeit der Random Walks auf einem Datensatz der Mäusebackenzähne 3.5.3 zu analysieren. Insgesamt ergab sich eine starke Skalierbarkeit der Random Walks, wobei der Speedup 87 % des theoretischen Maximums für 7 CPUs, 77 % für 14 CPUs und 68 % für 28 CPUs betrug (Abb. 21c). Multiprocessing mit mehreren GPUs wird bei den GPU-basierten Random Walks mit OpenMPI realisiert. Skalierbarkeitstests wurden mit 4 NVIDIA Tesla V100 sowie mehreren Datensätzen durchgeführt (Abb. 21b). Die Methode skaliert gut mit einer steigenden Anzahl von GPUs. Bei 4 GPUs reicht die Beschleunigung von 46 % (Theropodenklaue 3.5.8) bis 71 % (Kopf der Bulldoggenameisenkönigin 3.5.9) des theoretischen Maximums. Im Durchschnitt skaliert das Verfahren hier mit 60 % des theoretischen Maximums.

3.6 Unsicherheit der Segmentierung

Durch die Bestimmung der Unsicherheit mit der die Segmentierung erstellt wurde, erhält der/die Anwender/-in ein Feedback über die Qualität des Ergebnisses. Durch diese Unsicherheit können Bereiche hervorgehoben werden, die manuell korrigiert oder deren vorsegmentierte Schichten überarbeitet werden sollten, bevor der Prozess anschließend erneut gestartet wird. Die Unsicherheit entsteht durch den Einfluss schlecht vorsegmentierter Schichten auf das Ergebnis (Abschnitt 3.7.1 & 3.8.1), aber auch durch problematische Bildbereiche, wie filigrane Strukturen, einem niedrigen Kontrast oder Bildartefakten. Das Segmentierungsergebnis gilt als unsicher, wenn Random Walks unterschiedlicher Label mit-

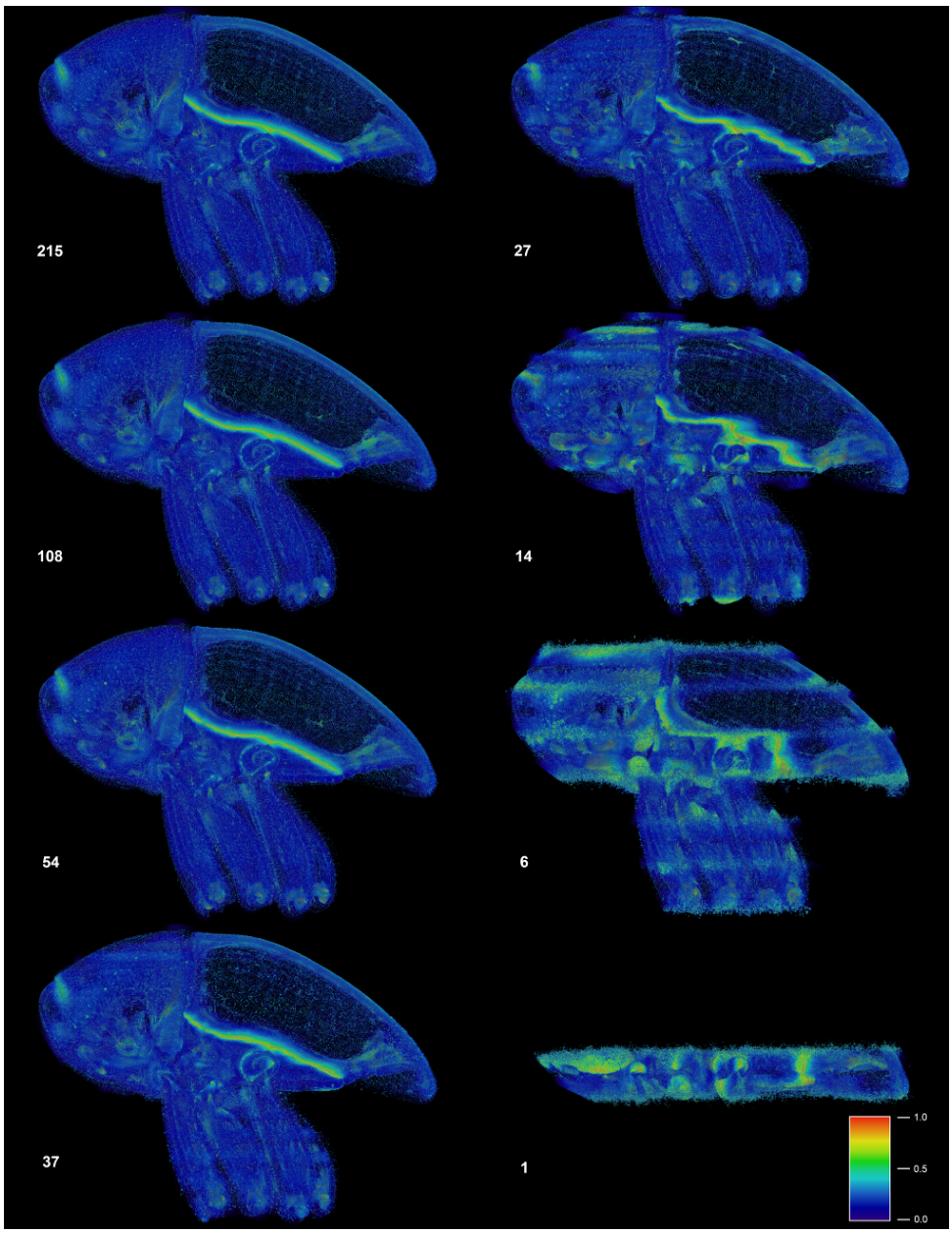

Abbildung 22: Unsicherheit der Segmentierung eines *Trigonopterus* Rüsselkäfers basierend auf einer unterschiedlichen Anzahl vorsegmentierter Schichten. Für 215, 108, 54 und 37 vorsegmentierte Schichten weisen die Ergebnisse ungefähr den gleichen Grad an Unsicherheit auf, während für 27 oder weniger Schichten die Unsicherheit signifikant zunimmt. Die auffällig helle Linie stellt die Grenze zwischen Elytra und Thorax dar, die eng miteinander verzahnt sind. Die Grenze ist im tomographischen Scan fast unsichtbar, was zu einer hohen Unsicherheit des Ergebnisses führt. Die Werte reichen von 0 (blau) bis 1 (rot), wobei 0 keine Unsicherheit und 1 eine sehr hohe Unsicherheit bedeutet. Die Volumen-Renderings wurden in Amira 2019.2 erstellt [Lösel et al., 2020].

einander konkurrieren, ohne dass dabei eine klare Zuordnung zu einem der Label erfolgen kann, das heißt, wenn die Anzahl der Treffer ungefähr gleich ist. Umgekehrt wird die Wahrscheinlichkeit für eine korrekte Zuordnung des Voxels als hoch angesehen, wenn die Anzahl der Treffer von Random Walks, die von einem Label ausgehen, deutlich gegenüber den anderen überwiegt oder wenn das Voxel sogar nur von Random Walks eines einzigen Labels getroffen wird. Die Unsicherheit der Segmentierung an der Stelle x lässt sich durch

$$U(x) = 1 - \prod_{i \neq \max} \left(1 - \frac{\Phi_i(x)}{\Phi_{\max}(x)}\right),$$

bestimmen, wobei $\Phi_1, \ldots, \Phi_n$ die Anzahl der Treffer der von n verschiedenen Label aus startenden Random Walks angibt. $\Phi_{\max}$ ist die maximal erreichte Anzahl an Treffern von einem Label. Die Unsicherheit nimmt Werte zwischen 0 und 1 an, wobei 0 keine Unsicherheit und 1 eine sehr hohe Unsicherheit bedeutet. Die Unsicherheit ist sehr hoch, wenn ein Voxel mindestens zwei Label mit der gleichen Wahrscheinlichkeit zugeordnet werden kann. Die Ergebnisse des *Trigonopterus* Rüsselkäfers zeigen eine Unsicherheit in der gleichen Größenordnung für eine Initialisierung mit 215, 108, 54 (was einer Vorsegmentierung jeder 5., 10. und 20. Schichten entspricht) und 37 an die Morphologie des Rüsselkäfers angepasste, vorsegmentierte Schichten (Abb. 22). Die Unsicherheit der Ergebnisse für 27 (Vorsegmentierung jeder 40. Schicht) oder weniger Schichten nimmt mit einer Reduzierung der Schichten deutlich zu. In allen Fällen wird die Grenze zwischen Elytra und Thorax, die eng miteinander verzahnt sind, in den Volumendarstellungen der Unsicherheit deutlich hervorgehoben. Die Grenze ist im tomographischen Scan fast unsichtbar (Abb. 23), was zu einer hohen Unsicherheit des Segmentierungsergebnisses führt.

3.7 Vergleich mit bestehenden Methoden

Um das Verfahren mit anderen gängigen Methoden zu vergleichen, wurde die Implementierung der GPU-basierten Random Walks in Biomedisa (siehe Kapitel 5) und die Implementierungen von RW (Abschnitt 2.2) in *scikit-image*, GC (Abschnitt 2.3) in *MedPy*, GeoS (Abschnitt 2.4) in *GeodisTK* sowie die konventionellen linearen Interpolationen von ITK und Amira 5.6 verwendet, um eine große Vielfalt biologischer und medizinischer Datensätze zu segmentieren (Tabelle 1 & 2, Abb. 20, 23, 24 & 25). Dabei wurde eine Computerarchitektur mit 2 Intel Xeon Gold 5120 CPU mit jeweils 14 Prozessorkernen, einer Grundfrequenz von 2.2 GHz und einer Boostfrequenz von 3.2 GHz sowie 750 GB RAM und 4 NVIDIA Tesla V100 Grafikprozessoren verwendet. Die Interpolation mit Amira war das einzige Verfahren, das nicht automatisiert auf dem Server durchgeführt werden konnte. Die Durchführung erfolgte daher durch einen/eine biologische(n) Experten/-in mithilfe eines Desktop-Computers.
Obwohl Biomedisa standardmäßig alle zur Verfügung stehenden Grafikprozessoren nutzt, wurde hier, für eine bessere Vergleichbarkeit, lediglich eine GPU verwendet. Während die Implementierungen von GC und GeoS sequenzielle Implementierungen sind, nutzt RW eine *multithreading BLAS* Bibliothek und automatisch mehrere Prozessorkerne zum Lösen

Tabelle 1: Quantitativer Vergleich verschiedener semi-automatischer Verfahren zur Segmentierung verschiedener Datensätze. Für die GPU-basierten Random Walks wurde die Implementierung in Biomedisa verwendet. Die jeweils höchste Genauigkeit ist hervorgehoben [Lösel et al., 2020].

Datensatz	**Methode**	**Dice (%)**	**ASD (Pixel)**	**Zeit (min)**
Fossile Wespe	Biomedisa	**95,98**	**0,458**	30
(56 Label, jede 20.	GeodisTK (GeoS)	94,21	0,583	332
Schicht vorsegmentiert,	Scikit-image (RW)	79,42	4,196	372
$1077 \times 992 \times 2553$ Voxel)	MedPy (GC)	88,29	3.803	1943
	ITK Interpolation	79,96	3,008	10
	Amira Interpolation	79,04	3,762	21
Trigonopterus	Biomedisa	**97,81**	**0,382**	20
(64 Label, adaptive	GeodisTK (GeoS)	96,99	0,553	415
Vorsegmentierung,	Scikit-image (RW)	80,79	6,001	494
keine Kreuzvalidierung,	MedPy (GC)	36,13	38 Label	2419
$1497 \times 734 \times 1117$ Voxel)	ITK Interpolation	82,75	4 Label	11
	Amira Interpolation	84,66	3 Label	13
Wespe in Bernstein	Biomedisa	**95,95**	**0,762**	48
(15 Label, jede 40.	GeodisTK (GeoS)	93,55	1,751	1057
Schicht vorsegmentiert,	Scikit-image (RW)	88,80	6,987	1815
$1417 \times 2063 \times 2733$ Voxel)	MedPy (GC)	90,80	8,136	5591
	ITK Interpolation	80,78	5,234	28
	Amira Interpolation	81,52	6,892	19
Theropodenklaue	Biomedisa	**88,67**	**0,409**	21
(1 Label, jede 80.	GeodisTK (GeoS)	60,02	3,845	182
Schicht vorsegmentiert,	Scikit-image (RW)	16,69	18,174	541
$1986 \times 1986 \times 3602$ Voxel)	MedPy (GC)	0,0	29,542	563
	ITK Interpolation	69,97	4,318	36
	Amira Interpolation	66,35	5,815	3
Medaka Skelett	Biomedisa	**84,24**	**1,210**	28
(1 Label, jede 10.–80.	GeodisTK (GeoS)	76,40	1,694	196
Schicht vorsegmentiert,	Scikit-image (RW)	5,01	17,122	537
$900 \times 1303 \times 4327$ Voxel)	MedPy (GC)	7,59	20,072	629
	ITK Interpolation	39,04	7,220	17
	Amira Interpolation	23,69	11,976	2
Menschliche Herzen	Biomedisa	**90,96**	0,715	3 s
(2 Label, jede 20. Schicht	GeodisTK (GeoS)	89,58	**0,655**	13 s
vorsegmentiert, durchschn.	Scikit-image (RW)	81,37	1,764	52 s
$157 \times 216 \times 167$ Voxel)	MedPy (GC)	88,36	1,021	61 s
	ITK Interpolation	70,14	3,468	1 s
	Amira Interpolation	68,67	3,862	< 30 s

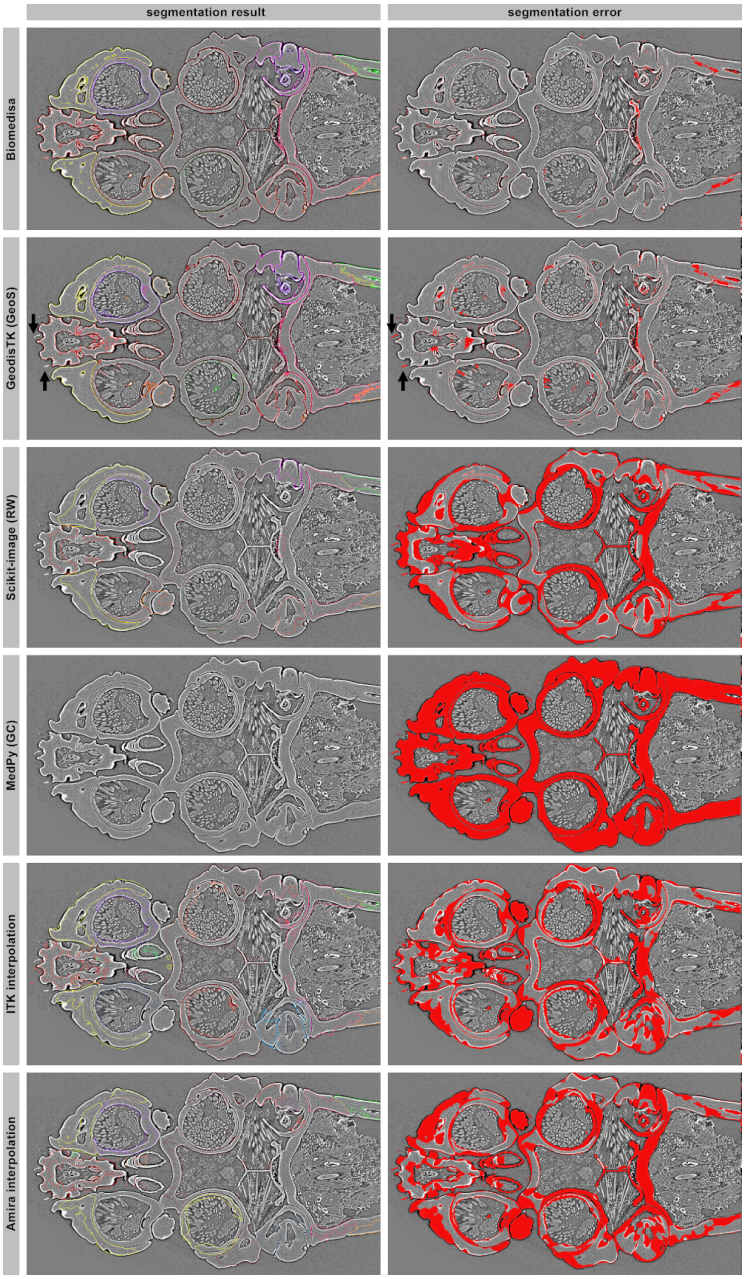

Abbildung 23: Visueller Vergleich der Segmentierungsergebnisse verschiedener semi-automatischer Verfahren zur Segmentierung eines *Trigonopterus* Rüsselkäfers. Zur Initialisierung der Verfahren wurde die, an die Morphologie des Rüsselkäfers angepasste, Vorsegmentierung verwendet (37 Schichten). Die angezeigten Segmentierungsfehler basieren auf den durch einen/eine Experten/-in erstellten Referenzschichten. Die GPU-basierten Random Walks (Biomedisa) konnten den Käfer bis auf die Grenze zwischen Elytra und Thorax akkurat rekonstruieren. Mit GeoS war es nicht möglich, die kleinen Haare am Kopf des Käfers (Pfeile) darzustellen. In der gezeigten Schicht fehlen große Teile des Käfers bei den Ergebnissen von RW, ITK und Amira. GC konnte in dieser Schicht nichts rekonstruieren [Lösel et al., 2020].

Tabelle 2: Quantitativer Vergleich verschiedener semi-automatischer Verfahren zur Segmentierung eines *Trigonopterus* Rüsselkäfers. Die Ergebnisse basieren auf einer unterschiedlichen Anzahl Label (**oben:** alle 64 Label, **unten:** 11 zusammengefasste Label) und einer abnehmenden Anzahl vorsegmentierter Schichten. Die jeweils höchste Genauigkeit ist hervorgehoben [Lösel et al., 2020].

		Jede 20. Schicht		**Jede 40. Schicht**		**Jede 80. Schicht**	
Label	**Methode**	**Dice (%)**	**Zeit (min)**	**Dice (%)**	**Zeit (min)**	**Dice (%)**	**Zeit (min)**
64	Biomedisa	**98,00**	25	**97,28**	15	**93,10**	10
	GeodisTK (GeoS)	97,50	427	95,91	433	91,57	462
	Scikit-image (RW)	91,64	388	79,44	507	45,44	648
	MedPy (GC)	58,99	2548	0,0	2584	0,0	2611
	ITK Interpolation	86,86	13	78,91	10	62,25	6
	Amira Interpolation	87,94	14	79,24	16	62,70	14
11	Biomedisa	**98,13**	24	**97,47**	14	**93,53**	8
	GeodisTK (GeoS)	97,66	143	96,14	144	92,26	153
	Scikit-image (RW)	91,82	198	79,74	234	45,83	285
	MedPy (GC)	60,60	742	0,0	720	0,0	778
	ITK Interpolation	86,29	17	78,21	14	58,55	11
	Amira Interpolation	85,60	4	75,52	3	54,10	3

der entstehenden linearen Gleichungssysteme 2.13. Um die linearen Gleichungssysteme in RW zu lösen, wird das konjugierte Gradientenverfahren in Kombination mit einem algebraischen Multigrid als Vorkonditionierer verwendet. Die Geodätischen Distanzen in GeoS werden mit dem Raster Scan Algorithmus 2.22 bestimmt.
Die Verwendung der Implementierungen von RW und GC werden sehr stark durch die Bildgröße begrenzt. Beide Verfahren lassen sich ohne Modifikation lediglich auf kleine Datensätze anwenden, wie beispielsweise auf die menschlichen Herzen, die menschlichen Kieferknochen oder die Mäusebackenzähne. Für die Segmentierung des *Trigonopterus* Rüsselkäfers ($1497 \times 734 \times 1117$ Voxel), oder noch größerer Datensätze, musste in dieser Arbeit eine spezielle blockweise Segmentierung entwickelt werden, bei der die Segmentierung zwischen zwei aufeinanderfolgenden vorsegmentierten Schichten erfolgt, um überhaupt einen Vergleich der Verfahren auf sehr großen volumetrischen Bilddaten zu ermöglichen. Zusätzlich zu diesen inneren Blöcken gibt es jeweils einen Block am oberen und unteren Ende der zu segmentierenden Objekte. In diesen zwei Blöcken erfolgt die Segmentierung durch eine Extrapolation über die jeweils erste bzw. letzte vorsegmentierte Schicht hinaus.
Obwohl GeoS in der Lage ist große Datensätze zu verarbeiten, wird auch hier der gleiche blockweise Ansatz eingesetzt. Dies erspart Rechenaufwand, da sich viele Segmente nur in einzelnen Teilbereichen befinden und sich nicht über das gesamten Volumen erstrecken. Somit ist es nicht erforderlich für alle Voxel im gesamten Volumen den kleinsten geodäti-

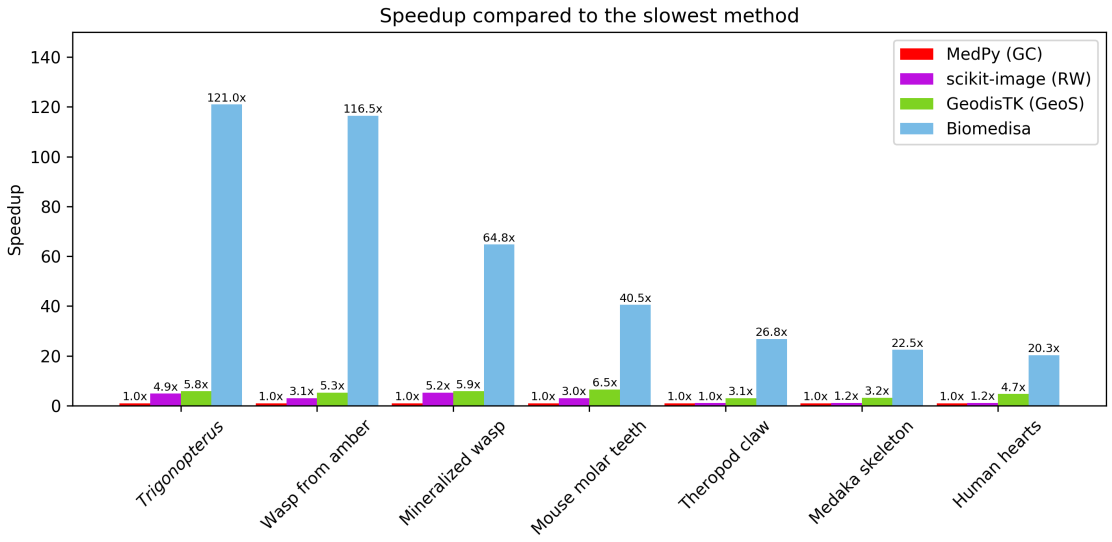

Abbildung 24: Speedup verschiedener semi-automatischer Verfahren gegenüber der jeweils langsamsten Methode. Die Werte basieren auf den Daten der Tabelle 1. Die Werte zu den Datensätzen Mäusebackenzähne und menschliche Herzen sind Durchschnittswerte [Lösel et al., 2020].

schen Abstand zu allen vorhandenen Label zu bestimmen, sondern es genügt sich auf die Label in dem jeweiligen Block zu beschränken.
Die Interpolationen von ITK und Amira können auch ohne diese blockweise Strategie verwendet werden, allerdings ist hier eine Extrapolation über die jeweils erste und letzte vorsegmentierte Schicht hinaus nicht möglich. Bis auf die Segmentierung mit Amira wurden alle Volumen vor der Verarbeitung automatisch auf den Bereich, der die Segmente enthält, zugeschnitten. Dabei wurde jeweils ein Abstand von mindestens 100 Voxel zur Vorsegmentierung eingehalten. Die Datensätze Medaka und Wespe in Bernstein waren zu groß, um sie mit einer GPU zu bearbeiten. Beide Volumen wurden in zwei sich überlappende Teilvolumen zerlegt, deren jeweilige Segmentierungen anschließend wieder zu einem einzigen Gesamtvolumen zusammengefasst wurden.
Die Segmentierungsergebnisse von RW und GC hängen sehr stark von der Wahl der Modellparameter β in der Gleichung 2.14 bzw. σ im Kantenterm 2.18 ab. Der *Raster-Scan*-Algorithmus zur Berechnung der geodätischen Distanzen (GeoS) benötigt lediglich die Angabe der durchzuführenden Iterationen i. Die Genauigkeit nimmt dabei mit jeder zusätzlichen Iteration zu, bis schließlich ein Plateau erreicht wird. Die Bearbeitungszeit steigt mit der Anzahl der Iterationen linear an. Das Verfahren der GPU-basierten Random Walks verhält sich ganz ähnlich. Auch hier gilt: Je höher die Anzahl der durchgeführten Random Walks pro vorsegmentiertes Pixel (*Number of Random Walks*, nbrw), desto höher die Genauigkeit und desto mehr Rechenzeit wird benötigt. Größere Distanzen zwischen den vorsegmentierten Schichten benötigen tendenziell mehr Schritte der Random Walks (*Steps of Random Walks*, sorw). Die Standardkonfiguration (norw = 10, sorw = 4000) wurde auf

Basis einer großen Anzahl verschiedener Anwendungsbeispiele empirisch gewählt. Um die Verfahren miteinander zu vergleichen wird jeweils die Standardkonfiguration gewählt (d. h. $\beta = 130$ (RW), norw $= 10$, sorw $= 4000$ (Biomedisa)). Gibt es keine Defaultwerte, werden die in den Beispielen der Dokumentation vorgeschlagenen Werte übernommen ($\sigma = 15$ (GC), $i = 4$ (GeoS)).
Der Vergleich wird auf einer Variation biologischer und medizinischer Datensätze, die durch verschiedene bildgebende Verfahren erzeugt wurden (MRT: Menschliche Herzen, μCT: Mäusebackenzähne, Medaka, CT: Menschliche Unterkiefer und SR-μCT: *Trigonopterus*, fossile Wespe, Wespe in Bernstein, Theropodenklaue), durchgeführt. Die Anzahl der segmentierten Objekte variiert dabei von einem Label (Theropodenklaue & Fauchschabe) bis zu 64 Label (*Trigonopterus*). Die Volumengrößen variieren von $157 \times 216 \times 167$ Voxel (Durchschnitt für die menschlichen Herzen) bis zu $1986 \times 1986 \times 3602$ Voxel (Theropodenklaue). Die Abstände zwischen den vorsegmentierten Schichten reichen von 10 Schichten bis zu 80 Schichten.
Um die Genauigkeit der Segmentierungsergebnisse zu bestimmen, werden zwei Metriken verwendet (Definition 3.24). Die erste Metrik, der Dice-Ähnlichkeitskoeffizient (Dice) gibt an, wie stark sich zwei Segmentierungen überlagern. Der Wert ist 0, wenn die gemeinsame Schnittmenge der Segmentierungen leer ist und 1, falls es eine perfekte Übereinstimmung gibt. Die zweite Metrik (*Average Surface Distance*, ASD) bestimmt den durchschnittlichen euklidischen Abstand eines Pixels auf dem Rand der Vorsegmentierung zum nächstgelegenen Punkt auf der Oberfläche des Segmentierungsergebnisses. Je kleiner diese Distanz, desto stärker stimmen Segmentierungsergebnis und Vorsegmentierung überein. Geht während des Segmentierungsprozesses mindestens ein Label verloren, kann die ASD nicht berechnet werden. Stattdessen wird hier die Anzahl der verlorenen Label angegeben. Beide gehören zu den am häufigsten eingesetzten Metriken zur Evaluation biomedizinischer Segmentierungsverfahren [Maier-Hein et al., 2018].

Definition 3.24. *Für zwei Segmentierungen X und X' mit $n \in \mathbb{N}$ verschiedenen Segmenten ist der Dice-Ähnlichkeitskoeffizient (Dice) definiert als*

$$Dice := \frac{2\sum_{i=1}^{n} |X_i \cap X_i'|}{|X| + |X'|},$$

wobei $|\cdot|$ der jeweiligen Anzahl an Voxel und X_i der Teilmenge von X mit Label i entspricht. Sei B die Menge der Segmenträndern in den vorsegmentierten Schichten und S' die Oberflächen des Segmentierungsergebnis, dann ist der durchschnittliche Abstand der Vorsegmentierung zur Oberfläche des Segmentierungsergebnis (Average Surface Distance, ASD) gegeben durch

$$ASD := \frac{1}{|B|} \sum_{i=1}^{n} \left(\sum_{p \in B_i} d(p, S_i') \right),$$

wobei

$$d(p, S_i') = \min_{p' \in S_i'} \|p - p'\|_2$$

dem euklidischen Abstand eines Punktes p auf dem Rand B_i der Vorsegmentierung mit Label i zum nächstgelegenen Punkt p' auf der entsprechenden Oberfläche S'_i des Segmentierungsergebnisses entspricht.

Der *Trigonopterus* Rüsselkäfer wurde in jeder 5. Schicht von einem/einer biologischen Experten/-in manuell vorsegmentiert. Insgesamt ergaben sich dadurch 215 vorsegmentierte Schichten. Die Segmentierungsergebnisse für die an die Morphologie angepasste Vorsegmentierung (37 Schichten, Tabelle 1) sowie für die Vorsegmentierungen jeder 20., 40. und 80. Schicht (was jeweils 54, 27 und 14 Schichten entspricht, Tabelle 2) wurden auf Basis der übrigen vorsegmentierten Schichten, die als Referenzschichten dienen, ausgewertet.
Für die Datensätze Medaka (hier nur das Skelett), Theropodenklaue, fossile Wespe, Wespe in Bernstein und Mäusebackenzähne wurde eine 2-fache Kreuzvalidierung verwendet. Die von biologischen Experten/-innen vorsegmentierten Schichten wurden dabei in zwei Gruppen A und B aufgeteilt, wobei A jede zweite vorsegmentierte Schicht enthielt und B die übrigen Schichten. Im ersten Schritt wurden die Schichten aus A genutzt, um die unterschiedlichen Verfahren zu initialisieren. Anschließend wurden die dadurch erzielten Ergebnisse mit den Schichten aus B verglichen und die Genauigkeit bestimmt. Im zweiten Schritt wurden die Schichten der Menge B zur Initialisierung und die Schichten von A zur Evaluation genutzt. Anschließend wurde der Durchschnitt beider erzielten Genauigkeiten berechnet (Tabelle 1).
Bei dieser Evaluation verdoppelten sich die Abstände zwischen den vorsegmentierten Schichten im Vergleich zur Berechnung der Ergebnisse in Abschnitt 3.5. Zum Beispiel wurde ursprünglich jede 10. Schicht der fossilen Wespe 3.5.7 manuell vorsegmentiert, die Evaluation beruht jedoch auf einer Vorsegmentierung jeder 20. Schicht. Die menschlichen Herzen 3.5.4 wurden ursprünglich vollständig von einem/einer klinischen Experten/-in segmentiert. Von dieser vollständigen Segmentierung wurde zunächst jede 10. Schicht extrahiert, um auch hier eine 2-fache Kreuzvalidierung durchführen zu können.
Alle Ergebnisse wurden ohne manuelle Nachbearbeitung evaluiert. Biomedisa erzielte dabei in allen Fällen die höchste Genauigkeit, gemessen anhand des Dice-Koeffizienten und hatte bis auf den Datensatz der menschlichen Herzen immer die kleinste ASD (Tabelle 1 & 2). Bei den Menschlichen Herzen war der durchschnittliche Abstand marginal größer gegenüber GeoS. Die jeweils benötigte Rechenzeit variierte beträchtlich. In allen Fällen war Biomedisa signifikant schneller als alle anderen Methoden, die ebenfalls die Bilddaten bei der Segmentierung berücksichtigen (Tabelle 1 & Abb. 24). Selbst die Rechenzeit der linearen Interpolationen von ITK und Amira bewegen sich in der Größenordnung der GPU-basierten Random Walks. Abhängig von der Volumengröße, der Anzahl der vorsegmentierten Schichten und der Anzahl der segmentierten Label, erreichte Biomedisa gegenüber GC ein mindestens 20-fach und bis zu 121-fach schnelleres Ergebnis (Abb. 24). Der Speedup gegenüber RW hatte einen Faktor zwischen 17 und 38. Der Speedup gegenüber GC betrug zwischen Faktor 4 und 22.

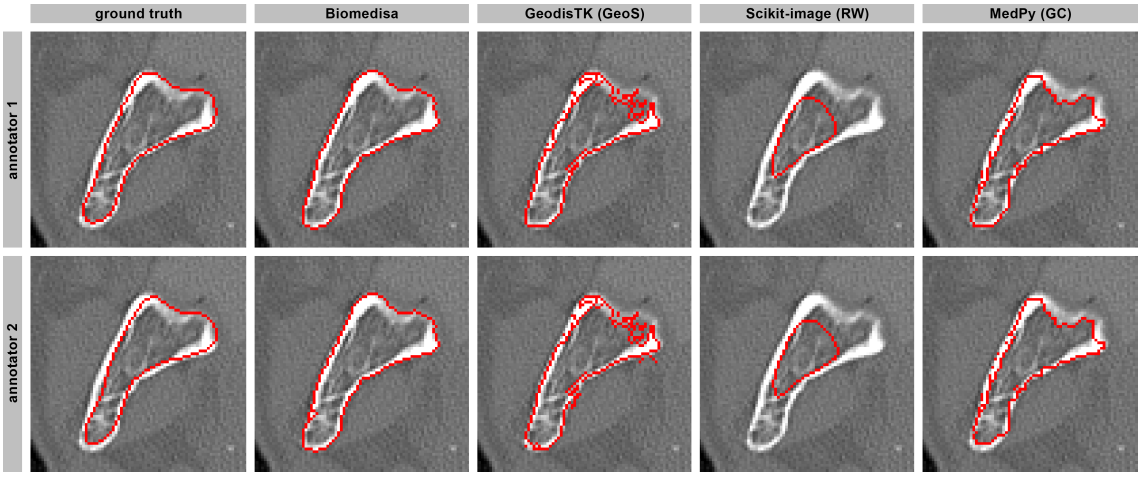

Abbildung 25: Visueller Vergleich der Robustheit gegenüber Initialisierungsfehlern. Ergebnisse verschiedener Segmentierungsmethoden zur Segmentierung eines menschlichen Unterkiefers basierend auf jeweils zwei fehlerbehafteten und inkonsistenten *Ground-Truth*-Daten, die durch zwei klinische Experten/-innen erstellt wurden [Wallner et al., 2019, Wallner et al., 2018]. Ungefähr jede 20. Schicht der manuellen Segmentierung wurde zur Initialisierung der Verfahren verwendet. Die Bilder zeigen das Segmentierungsergebnis in einer mittleren Schicht zwischen zwei vorsegmentierten Schichten [Lösel et al., 2020].

3.7.1 Robustheit gegenüber Initialisierungsfehlern

Um die Robustheit der verschiedenen Methoden gegenüber Fehlern bei der Vorsegmentierung zu testen, wurden 10 vollständig manuell segmentierte menschliche Mandibeln (Abschnitt 3.5.10) verwendet. Jeder Datensatz wurde von zwei klinischen Experten/-innen vollständig manuell segmentiert [Wallner et al., 2018], was jeweils zu zwei unterschiedlichen Segmentierungen führte (Abb. 25 Spalte 1). Etwa jede 20. Schicht dieser manuellen Segmentierungen diente der Initialisierung der verschiedenen Verfahren. Einige manuelle Segmentierungen weisen Schichten mit teilweise oder vollständig fehlenden Label auf. Eine äquidistante Wahl der vorsegmentierten Schichten war aus diesem Grund nicht immer möglich. Im Gegensatz zu GeoS, RW und GC zeigt Biomedisa eine weitaus höhere Robustheit gegenüber den ungenau vorsegmentierten Schichten (Abb. 25 Spalte 2-5). Darüber hinaus sind die Segmentierungsergebnisse von Biomedisa, die auf den beiden unterschiedlichen Initialisierungen basieren, einander ähnlicher (durchschnittlicher Dice-Score von 98,83 %) als die vorsegmentierten Schichten, die zur Initialisierung der Segmentierung herangezogen wurden (durchschnittlicher Dice-Score von 94,62 %). Das heißt, durch Biomedisa wurde eine Fehlerdämpfung erzielt. Im folgenden Abschnitt 3.8 wird eine Methode vorgestellt, um den Einfluss einer unsicheren Initialisierung auf das Segmentierungsergebnis visualisierbar und quantifizierbar zu machen.

3.8 Uncertainty Quantification

Viele Verfahren zur Segmentierung biomedizinischer Bilddaten sind durch physikalische Phänomene motiviert und können wie auch diese durch ein System von Differentialgleichungen modelliert werden. Zum Beispiel basiert das in Abschnitt 2.2 beschriebene kombinatorisches Dirichlet-Problem auf einer elliptischen partiellen Differentialgleichung

$$\nabla \cdot (D \nabla u) = 0, \tag{3.11}$$

$$u|_{\partial\Omega_1} = 1, \tag{3.12}$$

$$u|_{\partial\Omega_2} = 0, \tag{3.13}$$

die auch die stationäre Wärmeverteilung $u\colon \Omega \to \mathbb{R}^n$ in einem anisotropen Körper modelliert, die sich nach einiger Zeit einstellt, wenn dieser am Rand $\partial\Omega_1$ erwärmt und am Rand $\partial\Omega_2$ gekühlt wird. Das Tensorfeld $D\colon \Omega \to \mathbb{R}^{n\times n}$ beschreibt dabei die örtliche Temperaturleitfähigkeit des Körpers. Jede Modellierung stellt jedoch in der Regel eine Annäherung an das reale physikalische System dar und verursacht somit einen Modellfehler, zum Beispiel durch Vereinfachung komplexer Wechselwirkungen. Oft gibt es für diese Gleichungen auch keine analytische Lösung und sie müssen numerisch gelöst werden, wodurch ein zusätzlicher, numerischer Fehler entsteht, zum Beispiel durch eine endlichdimensionale Diskretisierung der kontinuierlichen Eigenschaften des Modells, um es auf einem Computer lösen zu können, oder durch die Verwendung von iterativen numerischen Lösern. Bei der sogenannten *Uncertainty Quantification* wird eine dritte Fehlerklasse betrachtet. Diese bezieht sich auf die Daten, die benötigt werden, um das Modell zu beschreiben, wie zum Beispiel Randbedingungen, Modellparameter oder die zu segmentierenden Bilddaten. Diese Daten sind häufig mit einer gewissen Unsicherheit versehen oder müssen sogar vollständig geschätzt werden. Beim kombinatorischen Dirichlet-Problem ist zum Beispiel der Tensor D in der Gleichung 3.11 abhängig von einem frei zu wählenden Modellparameter β. Die Vorsegmentierungen Ω_1 und Ω_2 in den Nebenbedingungen 3.12 und 3.13 sind in der Regel durch die Nutzer/-innen vorgegeben und oft fehlerbehaftet (Abb. 25). Aufgrund dieser unsicheren Systemparameter kann auch die Lösung $u = u(D, \Omega_1, \Omega_2)$ nicht eindeutig bestimmt werden. Werden diese Daten durch einen deterministischen Wert im Modell repräsentiert, vernachlässigt dies die Empfindlichkeit der Lösung gegenüber des eingebrachten Fehlers. Ersetzt man hingegen die Terme D, Ω_1 und Ω_2 durch Zufallsvariablen, so kann auch die Lösung u durch eine Zufallsvariable beschrieben werden. Auf diese Weise können Aussagen über die Sensitivität der Lösung gegenüber Variationen der Systemparameter erhalten werden. Das bedeutet, es kann bestimmt werden, wie wahrscheinlich bestimmte Ergebnisse sind, wenn einige Aspekte des Systems nicht genau bekannt sind. Von Interesse sind hierbei in erster Linie die stochastischen Momente, wie zum Beispiel der Mittelwert und die Varianz.

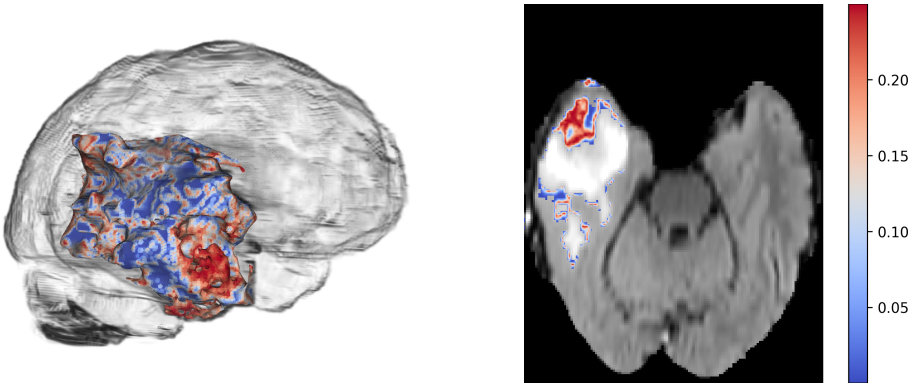

Abbildung 26: Unsichere Segmentierung eines Hirntumors aufgrund einer fehlerbehafteten Initialisierung. Die Abbildungen zeigen die Varianz der Segmentierung basierend auf einer leicht variierend vorsegmentierten Schicht in der Mitte des Tumors. **links** Die Unsicherheit verteilt sich über das gesamte Segment. Im vorderen unteren Bereich befindet sich ein größeres Areal, das durch eine hohe Varianz gekennzeichnet ist. **rechts** Darstellung einer unteren Schicht des MRT-Scans (T2-FLAIR) mit hoher Varianz in der Segmentierung.

3.8.1 Fehlerbehaftete Initialisierung

Die Initialisierung ist nicht nur für das kombinatorische Dirichlet-Problem, sondern für alle hier dargestellten halbautomatischen Verfahren ein wesentlicher Unsicherheitsfaktor, sei es aufgrund fehlender biologischer oder medizinischer Kenntnisse der zu segmentierenden Objekte oder aufgrund einer sich einschleichenden Ermüdung bei der Vorsegmentierung. Das Segmentierungsergebnis wird dabei sehr stark von der Genauigkeit der vorsegmentierten Schichten beeinflusst. Im Folgenden soll daher der Einfluss einer fehlerbehafteten Vorsegmentierung auf die Genauigkeit des GPU-basierten Segmentierungsverfahrens analysiert werden (Abb. 26). Dabei werden die Ränder der Vorsegmentierung an $N \in \mathbb{N}$ äquidistant verteilten Stützpunkten mittels Spline-Interpolation dargestellt, so dass sich durch Variation der Stützpunkte eine sich variierende Vorsegmentierung modellieren lässt. Da sich eine hinreichend genaue Interpolation der Kontur nur durch eine große Anzahl an Stützpunkten realisieren lässt ($N \gg 10$), wird zur Bestimmung der Varianz der Ergebnisse eine Monte-Carlo-Simulation durchgeführt. Da hierfür mehrere tausend Simulationsschritte x_i durchgeführt werden müssen, ist das Abspeichern der Zwischenergebnisse und das anschließende Berechnen der stochastischen Momente häufig nur für kleine volumetrische Bilddaten möglich. Für größere Datensätze kann der Erwartungswert durch das arithmetische Mittel

$$\bar{x} = \frac{1}{k} \sum_{i=1}^{k} x_i$$

geschätzt und dieses durch

$$M_{n+1} = \frac{1}{n+1}\sum_{i=1}^{n+1} x_i = \frac{x_1 + x_2 + \ldots + x_{n+1}}{n+1} \tag{3.14}$$

$$= \frac{nM_n + x_{n+1}}{n+1} \tag{3.15}$$

$$= \frac{(n+1)M_n + x_{n+1} - M_n}{n+1}$$

$$= M_n + \frac{x_{n+1} - M_n}{n+1},$$

für $n = 0, \ldots, k-1$ und $M_0 = 0$, rekursiv berechnet werden. Die Schätzung der Varianz erfolgt durch die Summe der quadratischen Abweichungen vom arithmetischen Mittel

$$S^2 = \sum_{i=1}^{k}(x_i - \bar{x})^2$$

und deren rekursive Berechnung durch

$$\begin{aligned}
S_{n+1}^2 &= \sum_{i=1}^{n+1}(x_i - M_{n+1})^2 = \sum_{i=1}^{n+1} x_i^2 - 2x_i M_{n+1} + M_{n+1}^2 \\
&= x_1^2 + x_2^2 + \ldots + x_{n+1}^2 - 2(x_1 + \ldots + x_{n+1})M_{n+1} + (n+1)M_{n+1}^2 \\
&\overset{3.14}{=} x_1^2 + x_2^2 + \ldots + x_{n+1}^2 - (n+1)M_{n+1}^2 \\
&= x_1^2 + x_2^2 + \ldots + x_n^2 - nM_n^2 + nM_n^2 + x_{n+1}^2 - (n+1)M_{n+1}^2 \\
&= S_n^2 + nM_n^2 + x_{n+1}^2 - (n+1)M_{n+1}^2 \\
&\overset{3.15}{=} S_n^2 + nM_n^2 + ((n+1)M_{n+1} - nM_n)^2 - (n+1)M_{n+1}^2 \\
&= S_n^2 + n(n+1)\left(M_{n+1} - M_n\right)^2,
\end{aligned}$$

für $n = 0, \ldots, k-1$ und $S_0^2 = 0$. Für die Segmentierung eines Hirntumors mit einer vorsegmentierten Schicht in der Mitte des Tumors (Abb. 26) ergaben sich für die Modellierung einer unsicheren Initialisierung mit $N = 50$ äquidistant verteilten Stützstellen $\vec{s}^{(i)} = \left(s_1^{(i)}, s_2^{(i)}\right)^T \in \mathbb{R}^2$ für $i = 1, \ldots, 50$ mit einer Standardabweichung von 2 Pixel entlang den Einheitsnormalen

$$\vec{n}_0^{(i)} = \frac{1}{\|\vec{n}^{(i)}\|_2}\vec{n}^{(i)},$$

mit

$$\vec{n}^{(i)} = \begin{pmatrix} n_1^{(i)} \\ n_2^{(i)} \end{pmatrix} \approx \frac{1}{2}\left[\begin{pmatrix} -s_2^{(i+1)} + s_2^{(i)} \\ s_1^{(i+1)} - s_1^{(i)} \end{pmatrix} + \begin{pmatrix} -s_2^{(i)} + s_2^{(i-1)} \\ s_1^{(i)} - s_1^{(i-1)} \end{pmatrix}\right]$$

und $\vec{s_0} \coloneqq \vec{s_{50}}$ sowie $\vec{s_{51}} \coloneqq \vec{s_1}$ bei $k = 10000$ durchgeführten Simulationen, durchschnittliche Dice-Scores (Definition 3.24) von 0.937 für die gestörte Initialisierung und 0.866 für

die Segmentierung. Da die ungestörte Segmentierung einen Dice-Score von 0.874 erreicht, bedeutet dies, dass das gestörte Segmentierungsergebnis lediglich 0.8 % an Genauigkeit verlor, wohingegen die Störung der Vorsegmentierung einen Genauigkeitsverlust von 6.3 % zur Folge hatte. Folglich konnte die starke Störung der Initialisierung durch die GPU-basierten Random Walks erheblich abgemildert und trotz Störung ein Ergebnis von annähernd gleicher Qualität erzielt werden. Der Datensatz entstammt den Trainingsbildern der *Brain Tumor Segmentation (BraTS) Challenge* Serie [Bakas et al., 2017, Menze et al., 2015, Baid et al., 2021]. Die *Ground-Truth*-Label wurden von einem/einer klinischen Experten/-in erstellt. Es wurde nur die T2-FLAIR-Sequenz verwendet (Abb. 26 rechts).

4 Deep Learning für eine automatisierte Segmentierung

Die Analyse einer großen Probenmenge kann geringfügige, aber statistisch und biologisch relevante Variationen in der Gehirn-Morphologie und Lateralisation (d. h. der funktionalen Aufgabenverteilung einzelner Gehirnregionen) zur Beantwortung ökologischer und evolutionsbiologischer Fragestellungen im Bereich der Neurowissenschaften aufdecken. Der manuelle Aufwand konventioneller Methoden (bspw. Histologische Schnitte, Scanning Elektronenmikroskopie oder Konfokalmikroskopie in Kombination mit einer manuellen Segmentierung) begrenzt jedoch die Anzahl der Proben, die für eine quantitative vergleichende Analyse von Insektengehirnen benötigt wird. Diese Einschränkung kann durch den Einsatz der nicht-invasiven Mikro-Computertomografie (μCT) in Kombination mit einer automatischen Segmentierung der Gehirnregionen überwunden werden. Die Entwicklungen im Bereich künstlicher neuronaler Netze haben in den letzten Jahren dazu geführt, dass deren Einsatz in einer zunehmenden Anzahl von Anwendungen herkömmlichen Ansätzen deutlich überlegen ist. In einigen Fällen haben künstliche neuronale Netze sogar menschliche Vergleichspersonen übertroffen, zum Beispiel in der Bildklassifizierung [He et al., 2016]. Insbesondere die Architektur der sogenannten *Convolutional Neural Networks* (CNNs) eignet sich hervorragend zur Analyse von Text-, Sprach- und Bildinformationen und ermöglicht die Implementierung effizienter Routinen, die große Mengen dieser Daten automatisiert verarbeiten. In diesem Kapitel wird daher zunächst die Approximationstheorie künstlicher neuronaler Netze erarbeitet (Abschnitt 4.1), um dann eine Netzwerkarchitektur zu beschreiben (Abschnitt 4.3.1), die eine automatische Segmentierung von 110 μCT-Scans von Honigbienengehirnen ermöglicht (Abschnitt 4.4.1).

4.1 Universalität künstlicher neuronaler Netze

Ein künstliches neuronales Netz besteht aus einer Eingabeschicht, einer oder mehreren verdeckten Schichten und einer Ausgabeschicht, wobei alle Neuronen einer verdeckten Schicht oder der Ausgabeschicht mit allen Neuronen der vorherigen Schicht verbunden sind (Abb. 27). Diese Netze werden daher häufig auch als *Fully Connected Neural Networks* (FCNNs) bezeichnet. Ein einfaches künstliches neuronales Netz $f_N\colon \mathbb{R}^d \to \mathbb{R}$ mit einer Zwischenschicht (bzw. verdeckten Schicht) und einer nichtlinearen Aktivierungsfunktion $\sigma\colon \mathbb{R} \to \mathbb{R}$ hat die Form

$$f_N(x) = \sum_{k=1}^{N} c_k \sigma(a_k^T x - b_k), \tag{4.1}$$

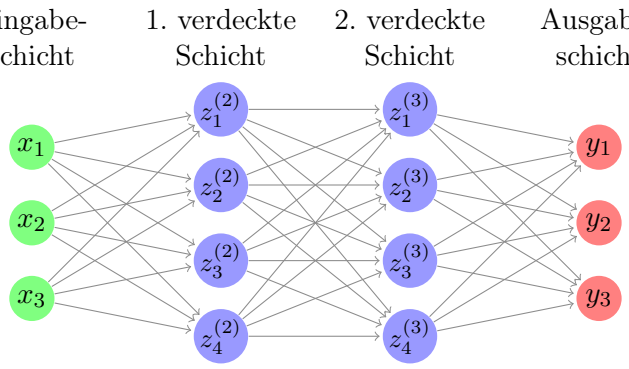

Abbildung 27: Künstliches neuronales Netz. Das hier dargestellte künstliche neuronale Netz besteht aus einer Eingabeschicht, zwei verdeckten Schichten und einer Ausgabeschicht. Alle Neuronen der verdeckten Schichten und der Ausgabeschichten sind jeweils mit allen Neuronen der vorherigen Schicht verbunden.

mit $a_k \in \mathbb{R}^d, b_k, c_k \in \mathbb{R}$. Beim sogenannten Training „lernt“ ein künstliches neuronales Netz die Eingaben $x \in \mathbb{R}^d$ auf die gewünschten Ausgaben $o(x)$ abzubilden. Typischerweise wird für eine Trainingsmenge X eine Kostenfunktion, zum Beispiel der mittlere quadratische Fehler der Netzausgaben

$$C = \frac{1}{|X|} \sum_{x \in X} \left(f_N(x) - o(x)\right)^2, \tag{4.2}$$

minimiert. Das Training wird in der Regel durch eine gradientenbasierte Optimierungstechnik, wie zum Beispiel dem stochastischen Gradientenabstieg (Abschnitt 4.2.1), realisiert. Der Begriff *Deep Neural Network* wurde eingeführt, um Netzwerke mit vielen Schichten von Netzwerken mit nur wenigen Schichten zu unterscheiden, wobei es keine klare Definition gibt, wie viele Schichten erforderlich sind, damit ein Netzwerk als „tief“ bezeichnet wird. Die mathematischen Grundlagen der Approximation von Funktionen durch Netzwerke der Form 4.1 wurden bereits in den späten 1980er Jahren entwickelt [Cybenko, 1989, Barron, 1993] und unmittelbar danach auf mehrschichtige Netzwerke erweitert [Hornik et al., 1989, Mhaskar, 1993]. Die Untersuchungen zeigten die Universalität künstlicher neuronaler Netze, das heißt, es konnte gezeigt werden, dass jede stetige Funktion f auf einer kompakten Teilmenge Ω von $\mathbb{R}^d$ für ein hinreichend großes N beliebig genau durch das entsprechende künstliche neuronale Netz f_N approximiert werden kann. Die Universalität wurde in [Cybenko, 1989, Hornik et al., 1989, Barron, 1993] für eine sigmoidale Aktivierungsfunktion σ gezeigt. Sigmoidal bedeutet: σ ist stetig, streng monoton steigend und es gilt $\lim_{x \to -\infty} \sigma(x) = 0$ und $\lim_{x \to \infty} \sigma(x) = 1$. Später konnte gezeigt werden, dass die Universalität für eine lokal beschränkte und stückweise stetige Aktivierungsfunktion σ sogar genau dann erfüllt ist, wenn σ kein Polynom ist [Leshno et al., 1993, Pinkus, 1999]. In den frühen 1990er Jahren wurden erste Studien bezüglich der Approximationsraten künstlicher neuronaler Netze veröffentlicht. Ist $\sigma \in C^\infty$ sigmoidal, $f = F|_{[-1,1]^d}$ für $F \in L^2(\mathbb{R}^d)$ und gilt für die Fourier-Transformierte $\hat{F}$, dass $|w|\hat{F}(w) \in L^1(\mathbb{R}^d)$, dann gilt

$\|f_N - f\|_{L^2_\mu([-1,1]^d)} = \mathcal{O}(1/\sqrt{N})$, wobei μ ein beliebiges Wahrscheinlichkeitsmaß ist [Barron, 1993]. Des Weiteren konnte für allgemeinere Aktivierungsfunktionen gezeigt werden, dass $\|f_N - f\|_{C([-1,1]^d)} = \mathcal{O}(N^{-s/d})$ für $f \in C^s([-1,1]^d)$ [Mhaskar, 1993].
Danach wurde es einige Zeit still um die künstlichen neuronalen Netze. Erst Anfang der 2010er Jahre erlebten sie eine Renaissance. Der Erfolg wurde katalysiert durch eine rasante Entwicklung der verwendeten Computerhardware und aufgrund immenser Datenmengen, die fortan zur Verfügung standen. Speziell der technische Fortschritt bei Grafikprozessoren (GPUs) erlaubte eine parallelisierte und damit effiziente Berechnung der für das Training von künstlichen neuronalen Netzen notwendigen Matrixmultiplikationen (Abschnitt 4.2.2). Daran anknüpfend gewann auch die mathematisch analytische Betrachtung wieder an Bedeutung. Ende der 2010er Jahre befassten sich etliche Studien mit den Approximationseigenschaften tiefer (das heißt vielschichtiger) künstlicher neuronaler Netze und betrachteten dabei den Einfluss der Anzahl an Schichten auf die Genauigkeit der Approximation [Telgarsky, 2016, Eldan and Shamir, 2016, Shaham et al., 2018, Petersen and Voigtlaender, 2018, Bölcskei et al., 2019, Yarotsky, 2017]. Schließlich wurde dann auch die spezielle Form der *Convolutional Neural Networks* (CNNs) analytisch betrachtet [Zhou, 2018, Zhou, 2020]. Dabei konnte gezeigt werden, dass die Anzahl der benötigten Schichten und der freien Parameter zum Erreichen einer bestimmten Approximationsgenauigkeit nicht mehr, wie bei einem *Fully Connected Neural Network*, exponentiell von der Dimension der Eingangsdaten abhängt, wodurch die empirische Erkenntnis, dass hochdimensionale Daten (wie Bilddaten) sehr effizient durch CNNs verarbeitet werden können, auf eine mathematisch analytische Grundlage gestellt wurde.
Historisch gesehen wurde die mathematische Struktur der künstlichen neuronalen Netze durch die Funktionalität des menschlichen Gehirns motiviert. Das erste künstliche neuronale Netz wurde in dem Versuch entwickelt, ein biologisches Neuron zu modellieren [McCulloch and Pitts, 1943]. Ein sogenanntes *McCulloch-* und *Pitts*-Neuron ist eine Funktion der Form

$$\mathbb{R}^d \ni x \mapsto \mathbb{1}_{\mathbb{R}^+}\left(\sum_{i=1}^{d} w_i x_i - t\right),$$

wobei $w_i, t \in \mathbb{R}$ für $i = 1, \dots, d$ und $\mathbb{1}_{\mathbb{R}^+} \colon \mathbb{R} \to \mathbb{R}$ ist die *Indikatorfunktion* mit

$$\mathbb{1}_{\mathbb{R}^+}(x) = \begin{cases} 0, & \text{falls } x < 0, \\ 1 & \text{sonst.} \end{cases} \tag{4.3}$$

Die nichtlineare Funktion $\mathbb{1}_{\mathbb{R}^+}$ entspricht einer sogenannten *Aktivierungsfunktion*. Die freien Parameter t und w_i heißen *Schwellenwert* und *Gewichte*. Für einen d-dimensionalen Eingabevektor $x \in \mathbb{R}^d$ bezeichnet man das Neuron als *aktiv* bzw. *feuernd*, falls $\mathbb{1}_{\mathbb{R}^+} = 1$ und als *inaktiv*, falls $\mathbb{1}_{\mathbb{R}^+} = 0$. Ein künstliches neuronales Netz lässt sich nun konstruieren, indem mehrere Neuronen zu jeweils einer Schicht zusammengefasst werden. Die Eingabe eines Neurons ergibt sich aus einer Gewichtung der Ausgaben der Neuronen der vorherigen Schicht. Ein einfaches Modell für ein solches Netzwerk ist das mehrschichtige Perzeptron (*Multilayer Perceptron*, MLP), vorgestellt von Rosenblatt [Rosenblatt, 1958]. Formal lässt sich ein mehrschichtiges Perzeptron wie folgt definieren:

Definition 4.1 (Multilayer Perceptron, MLP)**.** *Sei $d, L \in \mathbb{N}, L \geq 2$ und $\sigma\colon \mathbb{R} \to \mathbb{R}$ eine Aktivierungsfunktion, dann ist ein mehrschichtiges künstliches neuronales Netz mit d-dimensionalem Eingabevektor $x \in \mathbb{R}^d$ und L Schichten eine Funktion $F\colon K \to \mathbb{R}^m$ mit $K \subset \mathbb{R}^d$ kompakt und*

$$x \mapsto F(x) \coloneqq T_L(\sigma(T_{L-1}(\dots\sigma(T_1(x))\dots))),$$

wobei $T_l(x_{l-1}) = A_l x_{l-1} + b_l$ mit $A_l \in \mathbb{R}^{N_l \times N_{l-1}}, b_l \in \mathbb{R}^{N_l}, x_l = \sigma(T_l(x_{l-1})) \in \mathbb{R}^{N_l}$ und $N_l \in \mathbb{N}$ für $l = 1, \dots, L$ sowie $N_0 = d, N_L = m$ und $x_0 = x$. Die Aktivierungsfunktion σ wird dabei elementweise auf die Vektoren $T_l \in \mathbb{R}^{N_l}$ angewandt.

Diese Definition lässt somit nur Verbindungen zwischen benachbarten Schichten zu. Dabei kann jede beliebige Funktion als Aktivierungsfunktion σ gewählt werden. Ein so definiertes künstliches neuronales Netz stimmt mit der Netzwerkstruktur der ursprünglichen biologischen Motivation kaum noch überein. Beim Training eines MLPs, also bei der sukzessiven Anpassung der freien Parameter A_l und b_l, so dass bspw. die Kostenfunktion 4.2 für eine gegebene Trainingsmenge minimiert wird, spricht man bei einer großen Anzahl von Schichten, aufgrund der dadurch resultierenden Tiefe des Netzwerks, auch vom sogenannten *Deep Learning*. Die folgende analytische Untersuchung wird weder klären können, warum künstliche neuronale Netze sich in der praktischen Anwendung in vielen Bereichen gegenüber anderen Ansätzen als überaus erfolgreich erwiesen haben, noch wird sie ein explizites Verfahren liefern, wodurch sich im Einzelfall ein geeignetes Netz konstruieren ließe, das eine bestimmte Genauigkeit erreicht. Es wird sich jedoch zeigen, dass künstliche neuronale Netze zumindest in der Lage sind, bei einer hinreichend großen Anzahl von Neuronen jede stetige Funktion auf einer kompakten Menge beliebig genau zu approximieren.

Definition 4.2 (Sigmoidalität)**.** *Eine stetige Funktion $f\colon \mathbb{R} \to \mathbb{R}$ heißt sigmoidale Funktion, wenn gilt:*

$$f(x) = \begin{cases} 0 & \text{für } x \to -\infty, \\ 1 & \text{für } x \to \infty. \end{cases}$$

Neben der Indikatorfunktion 4.3 erfüllen viele weitere Funktionen diese Eigenschaft. In der Praxis relevant sind insbesondere die glatten Funktionen $0.5 \cdot \tanh(x) + 0.5$ sowie die auf der Exponentialfunktion beruhenden Funktion

$$\sigma_e(x) = \frac{1}{1 + e^{-x}}. \tag{4.4}$$

Definition 4.3 (Regularität)**.** *Ein Borel-Maß $\mu\colon \mathcal{A} \to [0, \infty)$ auf der σ-Algebra $\mathcal{A}$ heißt regulär, wenn für jedes $A \in \mathcal{A}$ gilt:*

$$\begin{aligned} \mu(A) &= \sup\{\mu(K) \mid K \subset A, K \text{ kompakt}\}, \\ &= \inf\{\mu(U) \mid A \subset U, U \text{ offen}\}. \end{aligned}$$

Lemma 4.4. *Sei $C(K)$ die Menge aller stetigen Funktionen auf $K \subset \mathbb{R}^d$ und*

$$\|f\|_\infty = \sup_{x \in K} |f(x)|$$

die Supremumsnorm. Wenn K kompakt ist, dann gilt:

$$\mathbb{B}_K = \{\mu \mid \mu \text{ ist ein signiertes reguläres Borel-Maß auf } K\}$$

ist der topologische Dual-Raum von $C(K)$.

Intuitiv weist ein Maß einer Menge eine Größe oder ein Volumen und jeder Teilmenge eine relative Größe im Verhältnis zur Gesamtmenge zu. Die folgenden Beweise beschränken sich auf reguläre Maße. Reguläre Maße nehmen auf einer kompakten Menge einen endlichen Wert an. Das ist insofern sinnvoll, da in einem metrischen Raum kompakte Mengen endlich und beschränkt sind und eine kompakte Menge somit keinen unendlichen Raum einschließen sollte.

Definition 4.5 (Universalität)**.** *Sei $\sigma\colon \mathbb{R} \to \mathbb{R}$ eine stetige Funktion, $d, L \in \mathbb{N}$ und $K \subset \mathbb{R}^d$ eine kompakte Teilmenge. Sei $\Sigma_L^d(\sigma)$ die Klasse von künstlichen neuronalen Netzen mit d-dimensionaler Eingabe, L Schichten, $N_L = 1$ und Aktivierungsfunktion σ. Dann heißt $\Sigma_L^d(\sigma)$ universal, wenn $\Sigma_L^d(\sigma)$ dicht in $C(K)$ liegt.*

Es zeigt sich, dass die folgende Eigenschaft der verwendeten Aktivierungsfunktion σ entscheidend dafür ist, ob für den Einheitswürfel $K = [0,1]^d$ die Menge $\Sigma_L^d(\sigma)$ dicht in $C([0,1]^d)$ liegt oder nicht.

Definition 4.6 (Diskriminator)**.** *Sei $d \in \mathbb{N}, K \subset \mathbb{R}^d$ kompakt, dann heißt eine stetige Aktivierungsfunktion σ diskriminatorisch, wenn für alle $a \in \mathbb{R}^d, b \in \mathbb{R}$ und für ein $\mu \in \mathbb{B}_K$ gilt:*

$$\int_K \sigma(a^T x + b)\, d\mu = 0 \quad \Longrightarrow \quad \mu = 0.$$

Satz 4.7. *Ist σ eine diskriminatorische Funktion, dann liegt $\Sigma_2^d(\sigma)$ dicht in $C([0,1]^d)$ bezüglich der Supremumsnorm. Das heißt, für jedes $f \in C([0,1]^d)$ und $\varepsilon > 0$ existiert eine Funktion $g(x) \in \Sigma_2^d(\sigma)$ mit*

$$\sup_{x \in [0,1]^d} |g - f| = \|g - f\|_\infty \leq \varepsilon.$$

Beweis. $\Sigma_2^d(\sigma)$ ist ein linearer Unterraum von $C([0,1]^d)$. Damit $\Sigma_2^d(\sigma)$ auch dicht in $C([0,1]^d)$ liegt, muss der Abschluss $\overline{\Sigma}_2^d(\sigma)$ mit $C([0,1]^d)$ übereinstimmen. Eine Folgerung des Satzes von Hahn-Banach besagt, dass für einen normierten Raum V ein echter Unterraum $U \subset V$ genau dann dicht in V liegt, wenn aus $x^* \in V^*$ und $x^*|_U = 0$ stets $x^* = 0$ folgt, wobei V^* der Dualraum von V ist. Angenommen $\Sigma_2^d(\sigma)$ liegt nicht dicht in $C([0,1]^d)$, dann ist $\overline{\Sigma}_2^d(\sigma)$ ein echter Unterraum von $C([0,1]^d)$. Nach der Folgerung des Satzes von Hahn-Banach gibt

es also ein $G^* \in C^*([0,1]^d)$ mit $G^* \neq 0$ und $G^*(\Sigma_2^d(\sigma)) = G^*(\overline{\Sigma}_2^d(\sigma)) = 0$. Nun besagt der Rieszsche Darstellungssatz, dass

$$G^*(h) = \int_{[0,1]^d} h(x)\, d\mu(x)$$

für ein endliches reguläres Borelmaß μ auf $[0,1]^d$ und alle $h \in C([0,1]^d)$. Somit gilt insbesondere auch für $h = \sigma(a^T x + b)$:

$$G^*(\sigma(a^T x + b)) = \int_{[0,1]^d} \sigma(a^T x + b)\, d\mu(x)$$

für alle $a \in \mathbb{R}^d, b \in \mathbb{R}$. Da $G^*(\Sigma_2^d(\sigma)) = 0$ gilt, folgt insbesondere

$$\int_{[0,1]^d} \sigma(a^T x + b)\, d\mu(x) = 0.$$

Da σ diskriminatorisch ist, muss somit $\mu = 0$ gelten. Dies widerspricht jedoch der Annahme, dass $G^* \neq 0$. Somit muss $\Sigma_2^d(\sigma)$ dicht in $C([0,1]^d)$ liegen. □

Dieser zentrale Satz in [Cybenko, 1989] zeigt, dass die diskriminatorische Eigenschaft der Aktivierungsfunktion die entscheidende Eigenschaft dafür zu sein scheint, dass das künstliche neuronale Netz die gewünschten Approximationseigenschaften aufweist. Im folgenden Lemma zeigt sich, dass die üblicherweise verwendeten sigmoidalen Aktivierungsfunktionen genau diese Eigenschaft aufweisen. Die Einschränkung auf das Intervall $[0,1]$ lässt sich ohne Weiteres auflösen und auf das Intervall $[-k,k]$ mit $k > 0$ erweitern, denn für jede Funktion $f(x) \in C([-k,k])$ mit $x \in \mathbb{R}^d$ und $k > 0$ gilt $f(2k(x - 0.5)) \in C([0,1])$.

Lemma 4.8. *Jede stetige sigmoidale Funktion $\sigma\colon \mathbb{R} \to \mathbb{R}$ ist diskriminatorisch.*

Beweis. Für alle $x, y \in K = [0,1]^d$ und $t, k \in \mathbb{R}$ gilt für $\sigma_\lambda(x) = \sigma(\lambda(y^T x + t) + k)$:

$$\lim_{\lambda\to\infty} \sigma_\lambda(x) = \gamma(x) = \begin{cases} 1 & \text{für } y^T x + t > 0, \\ 0 & \text{für } y^T x + t < 0, \\ \sigma(k) & \text{für } y^T x + t = 0. \end{cases}$$

Das heißt, $\sigma_\lambda(x)$ konvergiert punktweise gegen $\gamma(x)$. Des Weiteren sind die Funktionen der Folge $\{\sigma_\lambda\}$ alle messbar auf K und $\sigma_\lambda(x)$ wird von einer Funkion auf K majorisiert, da $|\sigma_\lambda(x)| \leq \max(1, \sigma(k))$ für alle λ und x. Somit folgt aus dem Satz von der dominierten Konvergenz:

$$\lim_{\lambda\to\infty} \int_K \sigma_\lambda(x)\, d\mu = \int_K \gamma(x)\, d\mu = \sigma(k)\mu(H_{a,t}^{(0)}) + \mu(H_{a,t}^{(+)}), \tag{4.5}$$

für alle a, t, k, mit der affinen Hyperebene $H_{a,t}^{(0)} = \{x \mid y^T x + t = 0\}$ und dem halb-offenen Raum $H_{a,t}^{(+)} = \{x \mid y^T x + t > 0\}$. Der Grenzwert 4.5 verschwindet, da nach Voraussetzung

$\int \sigma(y^T x + t)\, d\mu = 0$ für alle $y \in \mathbb{R}^d, t \in \mathbb{R}$ und damit auch $\int \sigma(\lambda(y^T x + t) + k)\, d\mu = \int \sigma_\lambda(x)\, d\mu = 0$. Es soll nun gezeigt werden, dass $\mu = 0$ gelten muss. Für ein fest gewähltes y und eine beschränkte, messbare Funktion h, sei

$$F(h) \coloneqq \int_K h(y^T x)\, d\mu,$$

ein lineares Funktional, wobei $F(h)$ beschränkt auf $L^\infty(\mathbb{R})$. Sei h die Indikatorfunktion mit

$$h(x) = \mathbb{1}_{[-t,\infty)}(x) = \begin{cases} 1, & \text{falls } x \geq -t, \\ 0, & \text{falls } x < -t. \end{cases}$$

Für $k \to \infty$ gilt:

$$F(h) = \int_K \mathbb{1}_{[-t,\infty)}(y^T x)\, d\mu = \mu(H_{a,t}^{(0)}) + \mu(H_{a,t}^{(+)}) = 0,$$

da $y^T x + t \geq 0$ für $y^T x \in [-t, \infty)$. Folglich gilt auch für alle Linearkombinationen von Indikatorfunktionen auf $[b_1, b_2]$:

$$0 = \int_K \mathbb{1}_{[b_1,\infty)}(y^T x)\, d\mu - \int_K \mathbb{1}_{[b_2,\infty)}(y^T x)\, d\mu = \int_K \mathbb{1}_{[b_1,b_2]}(y^T x)\, d\mu \tag{4.6}$$

und somit für jede einfache Funktion, i. e. Summe von Vielfachen von Indikatorfunktionen. Analog folgt für $k \to -\infty$, dass $F(h) = 0$ auf dem offenen Intervall $(-t, \infty)$. Da einfache Funktionen dicht in $L^\infty(\mathbb{R})$ liegen, folgt $F = 0$. Insbesondere gilt für die beschränkten, messbaren Funktionen $s(u) = \sin(m^T u)$ und $c(u) = \cos(m^T u)$:

$$F(c + is) = \int_K \cos(m^T x) + i \sin(m^T x)\, d\mu = \int_K e^{im^T x}\, d\mu = 0,$$

für alle m. Hieraus folgt, dass die Fourier-Transformierte von μ verschwindet und somit, dass μ selbst verschwindt. Woraus folgt, dass σ diskriminatorisch ist. □

4.1.1 ReLU-Aktivierungsfunktion

Neben der auf $\tanh(x)$ basierenden sigmoidalen Funktion und der speziellen Sigmoidfunktion 4.4 ist in der Praxis vor allem die sogenannte ReLU-Funktion (*Rectified Linear Unit*)

$$\mathrm{ReLU}(x) = \max\{x, 0\} = \begin{cases} x, & \text{falls } x \geq 0, \\ 0 & \text{sonst}, \end{cases} \tag{4.7}$$

als Aktivierungsfunktion relevant. Ein entscheidender Vorteil dieser Funktion gegenüber den zuvor genannten Aktivierungsfunktionen ist, dass sie nicht die Auswertung einer Exponentialfunktion benötigt und sich somit wesentlich schneller berechnen lässt. In der

Tat erfüllt auch die ReLU-Funktion die Eigenschaft diskriminatorisch zu sein, denn für $a \in \mathbb{R}, b_1 < b_2$ und

$$\begin{aligned} H_a(x) \coloneqq &\operatorname{ReLU}(ax - ab_1 + 1) - \operatorname{ReLU}(ax - ab_1) \\ &- \operatorname{ReLU}(ax - ab_2) + \operatorname{ReLU}(ax - ab_2 - 1) \end{aligned}$$

gilt für jedes $x \in \mathbb{R}$

$$H_a(x) \to \mathbb{1}_{[b_1, b_2]} \text{ für } a \to \infty,$$

denn

$$H_a(x) = \begin{cases} 0 & \text{für } x < b_1 - \frac{1}{a}, \\ a(x - b_1 + \frac{1}{a}) \leq 1 & \text{für } b_1 - \frac{1}{a} \leq x < b_1, \\ \operatorname{ReLU}(ax - ab_1 + 1) - \operatorname{ReLU}(ax - ab_1) = 1 & \text{für } b_1 \leq x < b_2, \\ 1 - \operatorname{ReLU}(ax - ab_2) = 1 - ax - ab_2 \leq 1 & \text{für } b_2 \leq x < b_2 + \frac{1}{a}, \\ 0 & \text{für } x \geq b_2 + \frac{1}{a}. \end{cases}$$

Dies bedeutet, dass Summen von ReLUs Indikatorfunktionen auf Intervallen punktweise approximieren können. Für

$$\int_K \operatorname{ReLU}(y^T x + t)\, d\mu = 0$$

für ein $\mu \in \mathbb{B}_K$ und für alle $y \in \mathbb{R}^d$ und $t \in \mathbb{R}$, folgt somit

$$\int_K \mathbb{1}_{[b_1, b_2]}(y^T x)\, d\mu = 0.$$

Die Fortsetzung des Beweises zu Lemma 4.8 an der Stelle 4.6 zeigt, dass auch die ReLU-Funktion diskriminatorisch ist.

4.1.2 Approximationsraten

Bisher konnte gezeigt werden, dass unter geringen Voraussetzungen an die Aktivierungsfunktion, jede stetige Funktion auf einer kompakten Menge $K \subset \mathbb{R}^d$ beliebig genau durch ein künstliches neuronales Netz mit zwei Schichten approximiert werden kann. Es lässt sich sogar zeigen, dass sich diese Approximationseigenschaft unter noch wesentlich schwächeren Annahmen einstellt und auf *Multilayer Perceptrons* (MLPs) erweitert werden kann [Leshno et al., 1993]. Insbesondere liegt die Klasse der MLPs mit lokal beschränkter, stückweise stetiger Aktivierungsfunktion genau dann dicht in $C(K), K \subset \mathbb{R}^d$, wenn die Aktivierungsfunktion kein Polynom ist. Allerdings geben weder die Approximationseigenschaft, noch deren Beweis einen Hinweis darauf, wie die Netzstruktur aussehen muss, damit eine gewisse Genauigkeit erreicht wird. In diesem Abschnitt wird daher ein Zusammenhang zwischen der Größe des Netzwerkes, d. h. der Anzahl der Neuronen und Schichten, und der Genauigkeit der Approximation hergestellt. Die Darstellung folgt in weiten Teilen [Petersen,

2020] und [Mhaskar, 1993]. Die grundlegende Idee ist es, die Approximationsfähigkeit eines künstlichen neuronalen Netzes auf die gut verstandene Approximation von Funktionen durch multivariate B-Splines zurückzuführen.

Definition 4.9 (Künstliches neuronales Netz)**.** *Ein künstliches neuronales Netz ist eine Folge von Matrix-Vektor-Tupel*

$$\Phi = ((A_1, b_1), \dots, (A_L, b_L)).$$

Für die Schichten $l = 1, \dots, L$ sind $A_l \in \mathbb{R}^{N_l \times N_{l-1}}$ die Gewichtsmatrizen zwischen zwei aufeinanderfolgenden Schichten, $b_l \in \mathbb{R}^{N_l}$ die sogenannten Bias-Vektoren und $N_l \in \mathbb{N}$ die Anzahl der Neuronen je Schicht, wobei $N_0 = d$ die Dimension der Eingangsdaten ist. Für ein künstliches neuronales Netz Φ und eine Aktivierungsfunktion $\sigma\colon \mathbb{R} \to \mathbb{R}$ sei die zugehörige Realisierung

$$\begin{aligned} &\mathrm{R}_\sigma(\Phi)\colon \mathbb{R}^d \to \mathbb{R}^{N_L}, \ \textit{mit} \\ &x \mapsto x_L \coloneqq R_\sigma(\Phi)(x) \end{aligned}$$

gegeben, wobei sich die Ausgabe $x_L \in \mathbb{R}^{N_L}$ aus der rekursiven Berechnung

$$\begin{aligned} x_0 &\coloneqq x, \\ x_l &\coloneqq \sigma(A_l x_{l-1} + b_l) \quad \textit{für } l = 1, \dots, L-1, \\ x_L &\coloneqq A_L x_{L-1} + b_L \end{aligned}$$

ergibt. Die Anwendung der Aktivierungsfunktion σ erfolgt dabei komponentenweise. Ein MLP (Definition 4.1) ist folglich eine Realisierung eines künstlichen neuronalen Netzes. Die Funktion $N(\Phi) \coloneqq d + \sum_{j=1}^{L} N_j$ heißt die Anzahl der Neuronen, $L(\Phi) \coloneqq L$ die Anzahl der Schichten und $M(\Phi) \coloneqq \sum_{j=1}^{L} M_j(\Phi)$ mit $M_j \coloneqq \|A_j\|_0 + \|b_j\|_0$ die Anzahl der freien Parameter des künstlichen neuronalen Netzes Φ, wobei $\|\cdot\|_0$ die Anzahl der Einträge verschieden von Null angibt.

Wenn man zeigen kann, dass ein künstliches neuronales Netz die Identität approximiert, kann man die Tiefe (d. h. die Anzahl der Schichten) eines künstlichen neuronalen Netzes beliebig erweitern, ohne dabei die Approximationseigenschaft des Netzwerkes zu verändern. In diesem Fall lassen sich zwei künstliche neuronale Netze Φ_1 und Φ_2, wobei die Dimension der Eingangsdaten von Φ_1 der Dimension der Ausgabedaten von Φ_2 entsprechen muss, zu einem neuen künstlichen neuronalen Netz $\Phi_1 \bullet \Phi_2$ mit

$$\mathrm{R}(\Phi_1 \bullet \Phi_2) \coloneqq \mathrm{R}(\Phi_1) \circ \mathrm{R}(\Phi_2) = \mathrm{R}(\Phi_1(\mathrm{R}(\Phi_2)),$$

verketten, ohne dabei die Eigenschaften aus Satz 4.7 zu verlieren. Ebenso lassen sich zwei künstliche neuronale Netze Φ_1 und Φ_2 mit $d_1, d_2 \in \mathbb{N}$ und $L_1 = L_2 \in \mathbb{N}$ durch eine sogenannte *Parallelisierung* entweder auf ein $(d_1 + d_2)$-dimensionales Netzwerk mit separierten Eingangsneuronen oder, falls $d_1 = d_2 = d$, auf ein d-dimensionales Netzwerk mit geteilten Eingangsneuronen erweitern. In beiden Fällen wird angenommen, dass es keine Verknüpfungen zwischen den daraus resultierenden, erweiterten Schichten $l = 1, \dots, L$ gibt. Des

Weiteren gilt für die Anzahl der Neuronen $N_l = N_{l_1} + N_{l_2}$ für $l = 1, \dots, L$. Da diese Netze immer auch eine Einschränkung von voll verknüpften Netzwerken sind, bei denen bestimmte Gewichte verschwinden, genügt es diese Netze zu betrachten.

Proposition 4.10. *Sei $d \in \mathbb{N}, K \subset \mathbb{R}^d$ kompakt und σ eine, auf einer offenen Umgebung von x differenzierbare, nicht konstante Aktivierungsfunktion, dann gibt es für jedes $\varepsilon > 0$ ein künstliches neuronales Netz $\Phi = ((A_1, b_1), (A_2, b_2))$ mit $A_1, A_2 \in \mathbb{R}^{d \times d}, b_1, b_2 \in \mathbb{R}^d$ und*

$$|\mathrm{R}(\Phi)(x) - x| < \varepsilon$$

für alle $x \in K$.

Beweis. Angenommen $d = 1$. $d > 1$ folgt dann durch Parallelisierung mit separierten Eingangsneuronen. Sei σ differenzierbar in einer ε-Umgebung von $x^* \in \mathbb{R}$ und $\sigma'(x^*) = \theta \neq 0$. Für $\lambda > 0$ sei

$$b_1 = x^*, \quad A_1 = \frac{1}{\lambda}, \quad b_2 = -\frac{\lambda\sigma(x^*)}{\theta} \quad \text{und} \quad A_2 = \frac{\lambda}{\theta},$$

dann gilt für alle $x \in K$:

$$|\mathrm{R}(\Phi)(x) - x| = \left| \lambda \frac{\sigma(\frac{x}{\lambda} + x^*) - \sigma(x^*)}{\theta} - x \right|.$$

Für $x = 0$ folgt unmittelbar, dass $|\mathrm{R}(\Phi)(x) - x| = 0$. Für $x \neq 0$ gilt:

$$|\mathrm{R}(\Phi)(x) - x| = \frac{|x|}{|\theta|} \left| \frac{\sigma(\frac{x}{\lambda} + x^*) - \sigma(x^*)}{\frac{x}{\lambda}} - \theta \right|.$$

Da $\frac{\sigma(\frac{x}{\lambda} + x^*) - \sigma(x^*)}{\frac{x}{\lambda}} \to \theta$ für $\lambda \to \infty$ gilt $|\mathrm{R}(\Phi)(x) - x| \to 0$ für alle $x \in K$. □

Definition 4.11 (Sigmoidalität der Ordnung q)**.** *Eine Funktion $\sigma \colon \mathbb{R} \to \mathbb{R}$ heißt sigmoidal der Ordnung $q \in \mathbb{N}$, falls $\sigma \in C^{q-1}(\mathbb{R})$ und*

$$\begin{aligned} &\frac{\sigma(x)}{x^q} \to 0 \text{ für } x \to -\infty, \\ &\frac{\sigma(x)}{x^q} \to 1 \text{ für } x \to \infty, \\ &|\sigma(x)| \lesssim (1 + |x|)^q \text{ für alle } x \in \mathbb{R}. \end{aligned}$$

Zum Beispiel ist die Funktion $x \mapsto \max(0, x)^q$ sigmoidal der Ordnung q und für $q = 1$ erhält man die ReLU-Funktion.

Definition 4.12 (B-Splines)**.** *Die Funktion*

$$\mathcal{N}_k(x) \coloneqq \frac{1}{(k-1)!}\sum_{l=0}^{k}(-1)^l\binom{k}{l}(x-l)_+^{k-1},\ \textit{für } x\in\mathbb{R},$$

auf $[0,k]$ *mit* $k\in\mathbb{N}$ *heißt univariater B-Spline der Ordnung* k*, wobei* $0^0\coloneqq 0$*. Für* $d,l\in\mathbb{N}, t\in\mathbb{R}^d$ *heißt*

$$\mathcal{N}_{l,t,k}^d(x)\coloneqq\prod_{i=1}^{d}\mathcal{N}_{l,t_i,k}(x_i),\ \textit{für } x=(x_1,\dots,x_d)\in\mathbb{R}^d,$$

multivariater B-Spline der Ordnung k *mit* $\mathcal{N}_{l,t_i,k}(x_i)\coloneqq\mathcal{N}_k(2^l(x_i-t_i))$*. Sei*

$$\mathcal{B}^k\coloneqq\{\mathcal{N}_{l,t_l,k}^d\mid t_l\in 2^{-l}\mathbb{Z}^d\}.$$

Der folgende Satz liefert eine Abschätzung der Genauigkeit der Approximation von Funktionen durch multivariate B-Splines.

Satz 4.13. *Sei* $k,d\in\mathbb{N}, p\in(0,\infty], 0<s<k$*. Dann gibt es eine Konstante* $C>0$*, so dass für jede Funktion* $f\in C^s\left([0,1]^d\right)$*, jedes* $\delta>0$ *und jedes* $N\in\mathbb{N}$ *Konstanten* $c_i\in\mathbb{R}$ *mit* $|c_i|\leq C\|f\|_\infty$ *und* $B_i\in\mathcal{B}^k$ *für* $i=1,\dots,N$ *existieren, so dass gilt:*

$$\left\|f-\sum_{i=1}^{N}c_iB_i\right\|_{L^p}\lesssim N^{\frac{\delta-s}{d}}\|f\|_{C^s}.$$

Insbesondere erzielt $\mathcal{B}^k$ *eine beste* N*-Term Approximationsrate der Ordnung* $N^{\frac{\delta-s}{d}}$ *für* $\mathcal{C}\coloneqq\{f\in C^s\left([0,1]^d\right)\mid\|f\|_{C^s}\leq 1\}$.

Dieser Satz findet sich in allgemeinerer Form in [Oswald, 1990]. Um nun Aussagen über die Approximation von Funktionen durch künstliche neuronale Netze zu erhalten, wird gezeigt, dass ein künstliches neuronales Netz jede Funktion $f\in\mathcal{B}^k$ beliebig genau approximieren kann.

Proposition 4.14 (Approximation multivariater B-Splines)**.** *Sei* $k,d\in\mathbb{N}, K>0$ *und* $\sigma\colon\mathbb{R}\to\mathbb{R}$ *sigmoidal der Ordnung* $q\geq 2$*. Dann gibt es eine Konstante* $C>0$*, so dass für jede Funktion* $f\in\mathcal{B}^k$ *und jedes* $\varepsilon>0$ *ein neuronales Netz* Φ_ε *existiert mit*

$$\begin{aligned}L(\Phi_\varepsilon)&=\lceil\log_2(d)\rceil+\lceil\max(\log_q(k-1),0)\rceil+1,\\ M(\Phi_\varepsilon)&=C,\\ \|f-\mathrm{R}(\Phi_\varepsilon)\|_{L^\infty([-K,K]^d)}&\leq\varepsilon.\end{aligned}$$

Beweis. Für $r\in\mathbb{N}$ sei $x_+^r\colon\mathbb{R}\to\mathbb{R}$ gegeben durch

$$x_+^r\coloneqq\begin{cases}x^r & \text{für } x\geq 0,\\ 0 & \text{für } x<0.\end{cases}$$

Der Beweis zeigt die Approximation eines kardinalen B-Spline $\mathcal{N}^d_{0,0,k}$ der Ordnung k. Der allgemeine Fall $\mathcal{N}^d_{l,t,k}$ folgt durch Skalierung und affine Transformation der Einträge A_1 und b_1 der ersten Schicht. Zunächst wird ein Netzwerk so konstruiert, dass es einen univariaten B-Spline approximiert. Hierfür ist es zunächst notwendig, die Funktion $x \mapsto x_+^{k-1}$ zu approximieren, aus der sich anschließend der gesuchte B-Spline aufbauen lässt. Da σ sigmoidal der Ordnung $q \geq 2$ gilt:

$$\left| a^{-q^T} \underbrace{\sigma \circ \sigma \circ \cdots \circ \sigma(ax)}_{T\text{-mal}} - x_+^{q^T} \right| \to 0 \text{ für } a \to \infty \text{ gleichmäßig für alle } x \in [-K', K'],$$

denn $\sigma(ax) \to (ax)^q, \sigma(\sigma(ax)) \to ((ax)^q)^q = (ax)^{q^2}$ und somit

$$\underbrace{\sigma(\cdots \sigma(\sigma}_{T\text{-mal}}(ax)) \cdots) \to (\cdots ((ax)^q)^q \cdots)^q = (ax)^{q^T} \text{ für } a \to \infty.$$

Sei $T \coloneqq \lceil \max(\log_q(k-1), 0) \rceil$, dann gilt $q^T \geq k-1$. Für jedes $K' > 0$ und jedes $\varepsilon > 0$ gibt es also ein künstliches neuronales Netz Φ^*_ε mit

$$\begin{aligned} L(\Phi^*_\varepsilon) &= \lceil \max(\log_q(k-1), 0) \rceil + 1, \\ |\mathrm{R}(\Phi^*_\varepsilon)(x) - x_+^p| &\leq \varepsilon, \end{aligned}$$

für jedes $x \in [-K', K']$ und $p \geq k-1$. Durch Verwendung des Vorwärtsdifferenzenquotienten folgt

$$\frac{\mathrm{R}\left(\Phi^*_{\delta^2}\right)(x+\delta) - \mathrm{R}\left(\Phi^*_{\delta^2}\right)(x)}{\delta} \to p x_+^{p-1} \text{ für } \delta \to 0.$$

Durch eine hinreichend häufige Wiederholung zeigt sich, dass für jedes $K' > 0$ und jedes $\varepsilon > 0$ ein künstliches neuronales Netz $\Phi^{\clubsuit}_\varepsilon$ existiert, so dass für alle $x \in [-K', K']$ gilt:

$$|\mathrm{R}(\Phi^{\clubsuit}_\varepsilon)(x) - x_+^{k-1}| \leq \varepsilon. \tag{4.8}$$

Es gibt also ein künstliches neuronales Netz $\Phi^{\heartsuit}_\varepsilon$, dessen Größe unabhängig von ε ist und das einen univariaten, kardinalen B-Spline bis auf eine Genauigkeit $\varepsilon > 0$ approximiert. Der nächste Schritt ist es, ein künstliches neuronales Netz zu konstruieren, das auf der Parallelisierung P von d univariaten Netzen $\Phi^{\heartsuit}_\varepsilon$ ohne geteilte Eingangsdaten basiert und die Einträge der d-dimensionalen Ausgabe miteinander multipliziert. Da σ der Ordnung $q \geq 2$ ist, folgt aus der Abschätzung 4.8 für $k = 3, q = 2, T = 1$ und $q^T = p = 2$, dass es für jedes $\varepsilon > 0$ und jedes $K' > 0$ ein künstliches neuronales Netz mit zwei Schichten gibt, das $f(x) = x_+^2$ für $x \in [-K', K']$ beliebig genau approximiert. Für $x = (x_1, x_2) \in \mathbb{R}^2$ gilt:

$$\begin{aligned} 2x_1x_2 &= (x_1 + x_2)^2 - x_1^2 - x_2^2 \\ &= (x_1 + x_2)_+^2 + (-x_1 - x_2)_+^2 - (x_1)_+^2 - (-x_1)_+^2 - (x_2)_+^2 - (-x_2)_+^2. \end{aligned}$$

Es existiert also ein künstliches neuronales Netz $\Phi_\varepsilon^{\#}$, das die Funktion $f(x_1, x_2) = x_1 x_2$ für $(x_1, x_2) \in [-K', K']^2$ beliebig genau approximiert. Der Einfachheit halber sei $\log_2(d) \in \mathbb{N}$ und

$$\Phi_\varepsilon^{\#,d,d/2} \coloneqq P(\underbrace{\Phi^{\#}, \dots, \Phi^{\#}}_{d/2\text{-mal}}).$$

Für alle $(x_1, \dots, x_d) \in [-K', K']^d$ gilt also:

$$\left|\mathrm{R}(\Phi_\varepsilon^{\#,d,d/2})(x_1, \dots, x_d) - (x_1 x_2, x_3 x_4, \dots, x_{d/2} x_d)\right| \leq \varepsilon.$$

Sei $(\cdot \bullet \cdot)$ die Verkettung zweier künstlicher neuronaler Netze, dann existiert für jedes $K' > 0$ und jedes $\varepsilon' > 0$ ein $\varepsilon > 0$ sowie ein künstliches neuronales Netz

$$\Phi_\varepsilon^{\#,d,1} \coloneqq \Phi_\varepsilon^{\#} \bullet \Phi_\varepsilon^{\#,4,2} \bullet, \dots, \bullet \Phi_\varepsilon^{\#,d,d/2}$$

von fester Größe mit

$$\left|\mathrm{R}(\Phi_\varepsilon^{\#,d,1})(x_1, \dots, x_d) - x_1 x_2 \cdots x_d)\right| \leq \varepsilon'.$$

Aufgrund der Konstruktion hat $\Phi_\varepsilon^{\#,d,1}$ insgesamt $\log_2(d) + 1$ Schichten. Die Verknüpfung der Parallelisierung P zur Approximation von d univariaten, kardinalen B-Splines mit dem Netz zur Multiplikation der jeweiligen Ausgaben, ergibt das gesuchte künstliche neuronale Netz

$$\Phi_\varepsilon \coloneqq \Phi_\varepsilon^{\#,d,1} \bullet P(\underbrace{\Phi_\varepsilon^{\heartsuit}, \dots, \Phi_\varepsilon^{\heartsuit}}_{d\text{-mal}}).$$

Insgesamt ergibt sich damit eine Anzahl von $\lceil \max(\log_q(k-1), 0) \rceil + \log_2(d) + 1$ Schichten für Φ_ε, wobei die Größe aller Komponenten unabhängig von ε ist. Für hinreichend kleines ε, approximiert Φ_ε den kardinalen B-Spline $\mathcal{N}_{0,0,k}^d$ nach Konstruktion beliebig genau auf $[-K, K]^d$. □

Zusammen mit [Petersen, 2020, Theorem 2.14 S. 9] ergibt sich aus Satz 4.13 und Proposition 4.14 die nachstehende Folgerung, die Aufschluss über die Approximationsrate von künstlichen neuronalen Netzen zur Approximation von Funktionen gibt.

Folgerung 4.15 (Approximation glatter Funktionen I)**.** *Sei $d \in \mathbb{N}, s > \delta > 0, p \in (0, \infty]$ und $\sigma \colon \mathbb{R} \to \mathbb{R}$ sigmoidal der Ordnung $q \geq 2$. Dann gibt es eine Konstante $C > 0$, so dass für jede Funktion $f \in C^s([0,1]^d)$ mit $\|f\|_{C^s} \leq 1$ und jedes $\frac{1}{2} > \varepsilon > 0$ ein künstliches neuronales Netz Φ existiert mit*

$$\begin{aligned} L(\Phi) &= \lceil \log_2(d) \rceil + \lceil \max(\log_q(\lceil s \rceil - 1), 0) \rceil + 1, \\ M(\Phi) &\leq C \varepsilon^{-\frac{d}{s-\delta}}, \\ \|f - \mathrm{R}(\Phi)\|_{L^p} &\leq \varepsilon. \end{aligned}$$

Folglich hängt die Netzwerkgröße von der stetigen Differenzierbarkeit der zu approximierenden Funktion ab. Je häufiger f stetig differenzierbar ist, desto weniger Neuronen werden für eine bestimmte Genauigkeit benötigt. Andererseits werden umso mehr Schichten benötigt, je größer s im Vergleich zu q ist. Allerdings ist dieser Zusammenhang der Form des Beweises geschuldet. Denn mit wesentlich mehr Aufwand lässt sich der folgende Satz zeigen, der nun auch die ReLU-Funktion als Aktivierungsfunktion einsetzt.

Satz 4.16 (Approximation glatter Funktionen II)**.** *Sei $d \in \mathbb{N}, s > \delta > 0, p \in (0, \infty]$ und $\sigma \colon \mathbb{R} \to \mathbb{R}$ die ReLU-Aktivierungsfunktion. Dann gibt es eine Konstante $C > 0$, so dass für jede Funktion $f \in C^s([0,1]^d)$ mit $\|f\|_{C^s} \leq 1$ und jedes $\frac{1}{2} > \varepsilon > 0$ ein künstliches neuronales Netz Φ existiert mit*

$$\begin{aligned} L(\Phi) &\leq C \log_2(1/\varepsilon), \\ M(\Phi) &\leq C\varepsilon^{-\frac{d}{s-\delta}}, \\ \|f - \mathrm{R}(\Phi)\|_{L^p} &\leq \varepsilon. \end{aligned}$$

Beweis. [Petersen, 2020, Theorem 3.19 S. 31] □

Sowohl in Folgerung 4.15 als auch in Satz 4.16 hängt die Approximationsordnung direkt von der Dimension d der Eingangsdaten ab. Zum Erreichen einer Approximation mit Genauigkeit $\varepsilon > 0$, steigt in beiden Fällen die Anzahl der benötigten freien Parameter mit größer werdender Dimension exponentiell an $\left(\mathcal{O}(\varepsilon^{-d/s})\right)$. Wobei dies natürlich nur eine obere Schranke ist, was im Einzelfall nicht ausschließt, dass kleinere Netzwerke existieren, die die gleiche Approximationsrate erreichen. Dieses häufig als „Fluch der Dimensionen“ bekannte Phänomen, ist nicht nur typisch für künstliche neuronale Netze, sondern auch für viele klassische Approximationstechniken. Betrachtet man jedoch die d-dimensionale Funktion $f(x) = \sum_{i=0}^{d} g_i(x_i)$, die sich aus d eindimensionalen Funktionen g_i aufbauen lässt, so benötigt man nur noch $d\mathcal{O}(\varepsilon^{-1/s})$ freie Parameter und damit deutlich weniger für $\varepsilon \to 0$. Man kann also annehmen, dass sich hochdimensionale Funktionen, die sich aus niedrigdimensionalen Funktionen aufbauen lassen, wesentlich effizienter approximieren lassen, als solche deren Struktur keine derartige Zerlegung zulässt. Diese Erkenntnis ebnet den Weg zu den sogenannten *Convolutional Neural Networks* (CNNs) deren Approximationsgenauigkeit nicht mehr wie bei einem *Fully Connected Neural Network* exponentiell von der Dimension der Eingangsdaten abhängt [Zhou, 2020]. Die Netzwerkstruktur der CNNs ist die Grundlage für viele der erfolgreichsten Anwendungen mit künstlichen neuronalen Netzen. Auch das im Rahmen dieser Arbeit vorgestellte Modell zur automatisierten Segmentierung volumetrischer Bilddaten basiert auf einem CNN (Abschnitt 4.3.1).

4.1.3 Segmentierung als Klassifikationsproblem

Soll ein künstliches neuronales Netz die d-dimensionalen Elemente $x = (x_1, \ldots, x_d)$ einer Trainingsmenge X in m verschiedene Klassen $K = \{k_1, \ldots, k_m\}$ unterteilen, so kann dies durch eine m-dimensionale Ausgabeschicht realisiert werden. Für ein Trainingspaar (x, y) mit $y \in \mathbb{R}^m$ gilt bei einer Zugehörigkeit von x zur i-ten Klasse mit $1 \leq i \leq m$, dass $y_i = 1$ und $y_j = 0$ für alle $j \neq i$, d. h.

$$(y_1, \ldots, y_{i-1}, y_i, y_{i+1}, \ldots, y_m)^T = (0, \ldots, 0, 1, 0, \ldots, 0)^T.$$

Insbesondere kann dadurch die Segmentierung eines Bildes in m Klassen bzw. Segmente erfolgen, indem jedes Pixel einer Klasse zugeordnet wird. Die Realisierung des Netzes (Definition 4.1) ist eine stetig differenzierbare Funktion $a\colon [0,1]^d \to [0,1]^m$ mit $x \mapsto a(x)$. Der folgende Satz zeigt, dass für ein erfolgreich trainiertes Netz, die Ausgabewerte Bayes-Wahrscheinlichkeiten darstellen [Richard and Lippmann, 1991]. So ist a_i die Wahrscheinlichkeit, dass der Eingabevektor x der Klasse k_i angehört. Die Klassifikation kann somit erfolgen, indem x der Klasse mit der höchsten Wahrscheinlichkeit zugeordnet wird.

Satz 4.17. *Die Ausgabewerte $a_1, \ldots, a_m$ eines, basierend auf dem mittleren quadratischen Fehler, trainierten Netzes mit m Ausgabeklassen, entsprechen für den Eingabevektor $x \in \mathbb{R}^d$ Bayes-Wahrscheinlichkeiten:*

$$a_i = P(k_i|x), \quad i = 1, \ldots, m,$$

wobei $P(k_i|x)$ die bedingte Wahrscheinlichkeit ist, dass x zur Klasse k_i gehört.

Beweis. Für die Trainingspaare (x, y) mit $x \in \mathbb{R}^d$ und $y \in \mathbb{R}^m$ sind in einem vollständig trainierten Netz die Gewichte so gewählt, dass

$$\Delta = E\left(\sum_{i=1}^{m}(a_i(x) - y_i)^2\right) \tag{4.9}$$

minimal ist, wobei $E(\cdot)$ der Erwartungswert ist. Mithilfe der gemeinsamen Wahrscheinlichkeit $P(x, k_i)$ für die Eingabe x und die Klasse k_i, lässt sich der Fehler 4.9 schreiben als

$$\Delta = \int \sum_{j=1}^{m}\left(\sum_{i=1}^{m}(a_i(x) - y_i)^2\right) P(x, k_j)\, dx.$$

Durch Substitution von $P(x, k_j) = P(k_j|x)P(x)$ erhält man:

$$\begin{aligned}
\Delta &= \int \left(\sum_{j=1}^{m} \sum_{i=1}^{m} (a_i(x) - y_i)^2 P(k_j|x) \right) P(x)\, dx \\
&= \int \sum_{l=1}^{m} \left(\sum_{j=1}^{m} \sum_{i=1}^{m} (a_i(x) - y_i)^2 P(k_j|x) \right) P(x, k_l)\, dx \\
&= E\left(\sum_{j=1}^{m} \sum_{i=1}^{m} (a_i(x) - y_i)^2 P(k_j|x) \right) \\
&= E\left(\sum_{j=1}^{m} \sum_{i=1}^{m} \left[(a_i(x))^2 P(k_j|x) - 2a_i(x) y_i P(k_j|x) + y_i^2 P(k_j|x) \right] \right).
\end{aligned}$$

Da $\sum_{j=1}^{m} P(k_j|x) = 1$ gilt:

$$\begin{aligned}
\Delta &= E\left(\sum_{i=1}^{m} \left[(a_i(x))^2 - 2a_i(x) \sum_{j=1}^{m} y_i P(k_j|x) + \sum_{j=1}^{m} y_i^2 P(k_j|x) \right] \right) \\
&= E\left(\sum_{i=1}^{m} \left[(a_i(x))^2 - 2a_i(x) E\left(y_i|x\right) + E\left(y_i^2|x\right) \right] \right),
\end{aligned}$$

wobei $E\left(y_i|x\right)$ und $E\left(y_i^2|x\right)$ die bedingten Erwartungswerte von y_i und y_i^2 sind. Durch Erweiterung mit $\sum_{i=1}^{m} \left(E(y_i|x)\right)^2$ erhält man:

$$\begin{aligned}
\Delta &= E\left(\sum_{i=1}^{m} \left[(a_i(x))^2 - 2a_i(x) E\left(y_i|x\right) + \left(E(y_i|x)\right)^2 + E\left(y_i^2|x\right) - \left(E(y_i|x)\right)^2 \right] \right) \\
&= E\left(\sum_{i=1}^{m} \left[a_i(x) - E\left(y_i|x\right) \right]^2 \right) + E\left(\sum_{i=1}^{m} var(y_i|x) \right),
\end{aligned}$$

wobei $var(y_i|x)$ die bedingte Varianz von y_i ist. Da nur der erste Term abhängig von der Netzausgabe a_i ist, wird Δ genau dann minimal, wenn $a_i(x) = E(y_i|x)$. Da für die Trainingsdaten gilt, dass $y_i = 1$ bei einer Zugehörigkeit von x zur i-ten Klasse und $y_j = 0$ für $j \neq i$ folgt, Δ ist minimal, wenn

$$a_i(x) = E\left(y_i|x\right) = \sum_{j=1}^{m} y_i P(k_j|x) = P(k_i|x).$$

□

4.2 Backpropagation-Algorithmus

Für einen Trainingsdatensatz X, bestehend aus n Trainingsbildern $x \in \mathbb{R}^d$ und korrespondierenden Label $y \in \mathbb{R}^m$, seien Gewichte $w_{jk}^{(l)} \in \mathbb{R}$ für $l = 1, \ldots, L$ gesucht, die die Kostenfunktion

$$C = \frac{1}{2} \sum_{x \in X} \left(a^{(L)}(x) - y \right)^2 \tag{4.10}$$

minimieren, wobei

$$a_j^{(0)} = x_j \qquad \text{und} \qquad a_j^{(l)} = \sigma\left(z_j^{(l)} \right) \tag{4.11}$$

mit

$$z_j^{(l)} = \sum_k w_{jk}^{(l)} a_k^{(l-1)}. \tag{4.12}$$

Der Einfachheit halber wird hier auf einen Schwellenwert verzichtet. Die meisten Optimierungsmethoden benötigen den Gradienten von C bezüglich der Gewichte $w_{jk}^{(l)}$. Sei

$$\delta_j^{(l)} := \frac{\partial C}{\partial z_j^{(l)}}$$

die partielle Ableitung von C bezüglich der Neuronen $z_j^{(l)}$ der Schicht l (Abb. 27). Somit gilt für die partiellen Ableitungen von C bezüglich der Gewichte $w_{jk}^{(l)}$:

$$\frac{\partial C}{\partial w_{jk}^{(l)}} = a_k^{(l-1)} \frac{\partial C}{\partial z_j^{(l)}} = a_k^{(l-1)} \delta_j^{(l)}.$$

Somit lässt sich der Gradient von C durch die Ableitungen bezüglich der Schichten des Netzwerks ausdrücken. Die zusätzlich benötigten Terme $a_k^{(l-1)}$ lassen sich mit den Zuweisungen 4.11 & 4.12 für jedes Trainingsbild x rekursiv bestimmen.

Satz 4.18. *Die Ableitung der Kostenfunktion C bezüglich der letzten Schicht L ist gegeben durch*

$$\delta_j^{(L)} = \left(a_j^{(L)} - y_j \right) \sigma'\left(z_j^{(L)} \right), \quad j = 1, \ldots, m,$$

wobei σ' die Ableitung der Aktivierungsfunktion σ nach $z_j^{(L)}$ ist.

Beweis. Mithilfe der Kettenregel gilt für die Ableitung bezüglich der letzten Schicht:

$$\delta_j^{(L)} = \frac{\partial C}{\partial z_j^{(L)}} = \sum_k \frac{\partial C}{\partial a_k^{(L)}} \cdot \frac{\partial a_k^{(L)}}{\partial z_j^{(L)}}.$$

Aus der Definition der Kostenfunktion 4.10 folgt, dass alle Terme für $k \neq j$ verschwinden, da $z_j^{(L)}$ nur von $a_j^{(L)}$ abhängt. Folglich gilt:

$$\delta_j^{(L)} = \frac{\partial C}{\partial a_j^{(L)}} \cdot \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} = \left(a_j^{(L)} - y_j\right) \sigma'\left(z_j^{(L)}\right).$$

□

Satz 4.19. *Für die verdeckten Schichten $l = L-1, \ldots, 1$ gilt:*

$$\delta^{(l)} = \left(\left(w^{(l+1)}\right)^T \delta^{(l+1)}\right) \odot \sigma'\left(z^{(l)}\right),$$

wobei $\odot$ das Hadamard-Produkt beschreibt.

Beweis. Mithilfe der Kettenregel folgt:

$$\begin{aligned}
\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} &= \sum_k \frac{\partial C}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}, \\
&= \sum_k \delta_k^{(l+1)} \cdot \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}, \\
&= \sum_k \delta_k^{(l+1)} w_{kj}^{(l+1)} \sigma'\left(z_j^{(l)}\right).
\end{aligned}$$

□

4.2.1 Stochastic Gradient Descent

Kleine Änderungen der Kostenfunktion C lassen sich durch kleine Änderungen der Gewichte $w_{jk}^{(l)}$ ausdrücken:

$$\Delta C \approx \sum_{j,k,l} \frac{\partial C}{\partial w_{jk}^{(l)}} \Delta w_{jk}^{(l)}.$$

Für eine sogenannte *Lernrate* $\eta > 0$ sei

$$\begin{aligned}
&\Delta w := -\eta \nabla C, \\
\Rightarrow\ &\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2, \\
\Rightarrow\ &\Delta C \leq 0.
\end{aligned}$$

Somit können zur Minimierung der Kostenfunktion C die Gewichte $w_{jk}^{(l)}$ in jedem Iterationsschritt durch

$$w_{t+1} = w_t - \eta \nabla C \tag{4.13}$$

angepasst werden. Für ein hinreichend kleines η garantiert dieser Optimierungsprozess die, unter Umständen jedoch sehr langsame, Konvergenz zu einem lokalen Minimum. Allerdings ist C keine konvexe Funktion, sondern in der Regel eine raue Mannigfaltigkeit mit unzähligen lokalen Minima. Empirisch hat sich gezeigt, dass eine leichte Modifikation des Gradientenabstiegsverfahrens oft dazu in der Lage ist, ein für die Problemstellung hinreichend gutes Minimum zu finden. Dabei wird der Trainingsdatensatz X für jede Epoche zufällig in disjunkte, gleichgroße Teilmengen (*Batches*) unterteilt und jedes *Batch* $B \subset X$ einmal zur Approximation des Gradienten

$$\nabla C \approx \frac{\partial C_B}{\partial \mathbf{w}}$$

im Iterationsschritt 4.13 verwendet, wobei $C_B = \frac{1}{2} \sum_{x \in B} \left(a^{(L)}(x) - y\right)^2$. Eine Epoche ist abgeschlossen, sobald alle Teilmengen einmal verwendet wurden. Anschließend wird X erneut randomisiert unterteilt und eine weitere Epoche gestartet. Für ein künstliches neuronales Netz mit zwei verdeckten Schichten (Abb. 27) und ReLU-Aktivierungsfunktion (Definition 4.7) ergibt sich somit die im Programm 3 illustrativ dargestellte Implementierung in der Programmiersprache *Python*. Die Trainingsdaten X liegen dabei in einem Array der Größe $n \times (d + m)$ vor.

Programm 3: *Stochastic Gradient Descent* Implementierung in der Programmiersprache *Python*.

```
import numpy as np

def ReLU(z):
    a = np.copy(z)
    a[z<0]=0
    return a

def ReLU_del(z):
    a_del = np.copy(z)
    a_del[z<0]=0
    a_del[z>=0]=1
    return a_del

if __name__ == "__main__":
    # loop over epochs
    for e in range(epochs):

        # shuffle data
        np.random.shuffle(X)

        # iterate over all batches
        for k in range(0, n, batch_size):

            # get one batch
            x = np.transpose(X[k:k+batch_size, :d])
            y = np.transpose(X[k:k+batch_size, -m:])

            # forward propagation
            z_1 = np.dot(W1, x)
            a_1 = ReLU(z_1)
            a_1_del = ReLU_del(z_1)

            z_2 = np.dot(W2, a_1)
            a_2 = ReLU(z_2)
            a_2_del = ReLU_del(z_2)

            z_3 = np.dot(W3, a_2)
            a_3 = ReLU(z_3)
            a_3_del = ReLU_del(z_3)

            # backpropagation
            delta = a_3_del * (a_3 - y)
            C_del_3 = np.dot(delta, np.transpose(a_2))

            delta = a_2_del * np.dot(np.transpose(W3), delta)
            C_del_2 = np.dot(delta, np.transpose(a_1))

            delta = a_1_del * np.dot(np.transpose(W2), delta)
            C_del_1 = np.dot(delta, np.transpose(x))

            # update weights
            W1 -= eta * C_del_1
            W2 -= eta * C_del_2
            W3 -= eta * C_del_3
```

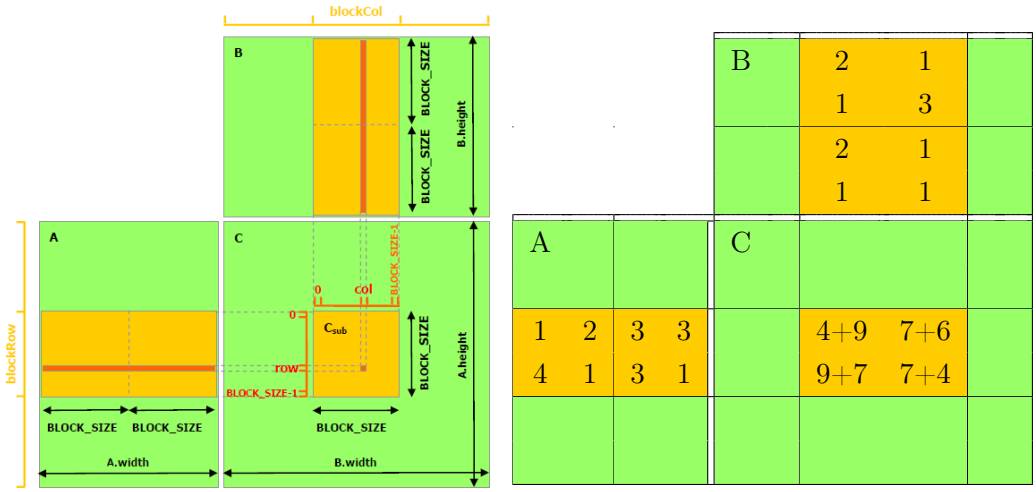

Abbildung 28: Blockweise Matrixmultiplikation mit einer GPU. Um die Submatrix Csub $= \left(\hat{c}_{ik}^{(2,2)}\right)$ zu erhalten, werden iterativ die Matrixmultiplikationen $\left(\hat{a}_{ij}^{(2,1)}\right) \cdot \left(\hat{b}_{jk}^{(1,2)}\right)$ und $\left(\hat{a}_{ij}^{(2,2)}\right) \cdot \left(\hat{b}_{jk}^{(2,2)}\right)$ durchgeführt und die Ergebnisse addiert [NVIDIA, 2021].

4.2.2 GPU-basierte Matrixmultiplikation

Die Zuweisung 4.12 und Satz 4.19 verdeutlichen, dass zur Optimierung der Gewichte in Programm 3 vorrangig Matrixmultiplikationen notwendig sind, die sich sehr effizient auf einer GPU berechnen lassen. Die Matrixmultiplikation zweier Matrizen $A = (a_{ij})$ und $B = (b_{jk})$ ist eine Abbildung

$$\cdot : \mathbb{R}^{l \times m} \times \mathbb{R}^{m \times n} \to \mathbb{R}^{l \times n}, \quad (A, B) \mapsto C = A \cdot B,$$

mit

$$C = (c_{ik}) \qquad \text{and} \qquad c_{ik} = \sum_{j=1}^{m} a_{ij} \cdot b_{jk},$$

für $i = 1, \ldots, l$ und $k = 1, \ldots, n$. Das bedeutet, dass jedes Element c_{ik} der Ergebnismatrix C die Summe der komponentenweisen Multiplikation der i-ten Zeile von A und der k-ten Spalte von B ist. Für die Realisierung der Matrixmultiplikation auf einer GPU ist zu beachten, dass Speicherzugriffe in der Regel wesentlich zeitaufwändiger sind als die Multiplikation zweier Zahlen, was eine sehr schnelle Operation darstellt. Bei der Matrixmultiplikation wird auf jedes Element a_{ij} insgesamt n-mal und auf jedes b_{jk} insgesamt l-mal zugegriffen. Für eine effiziente Implementierung kann die Speicherhierarchie moderner GPUs genutzt werden. GPUs haben verschiedene Speichertypen mit unterschiedlichen Eigenschaften: Register, Local Memory, Shared Memory, Constant Memory, Texture memory und Global Memory [Cheng et al., 2014, S. 138]. Während auf den Registerspeicher und den Local Memory nur ein einziger Thread zugreifen kann, können alle Threads eines

Blocks auf die Daten im Shared Memory zugreifen. Darüber hinaus können die Threads eines Blocks ihre Berechnungen synchronisieren. Auf den Constant und Global Memory haben alle Threads gleichermaßen Zugriff, jedoch ist der Zugriff auf den gemeinsam genutzten Speicher (Shared Memory) um einige Größenordnungen schneller als der Zugriff auf den globalen Speicher. Im Vergleich zum Global Memory hat der Shared Memory zum Beispiel auf der Fermi und Kepler-Architektur eine 20 bis 30-mal geringere Latenz und eine um mehr als 10-mal höhere Bandbreite [Cheng et al., 2014, S. 216].
Die natürliche blockweise Hardwarearchitektur der GPUs kann verwendet werden, um die Geschwindigkeitsvorteile des Shared Memorys für die Multiplikation zweier Matrizen auszunutzen (Programm 4). Dabei werden die Elemente von mehreren Threads parallel in den Shared Memory geladen und können dann von allen anderen Threads des jeweiligen Blocks sehr schnell weiterverarbeitet werden. Jedes Element der Matrix C wird dabei einem Thread in einem Block der Größe $s \times s$ zugeordnet. Der Einfachheit halber wird hier angenommen, dass l, m, n Vielfache von s sind und die Blöcke quadratisch sind. Alle Blöcke sind somit in einem Grid der Größe $o \times q$ angeordnet, wobei $s \cdot o = l$ und $s \cdot q = n$. Jeder Block berechnet eine Submatrix $\left(\hat{c}_{ik}^{(r,u)}\right) \in \mathbb{R}^{s \times s}$ für $r = 1, \dots o$ und $u = 1, \dots q$ (Abb. 28), wobei jeder Thread ein Element dieser Submatrix berechnet (Zeile 41). Hierfür durchläuft der Block alle Submatrizen $\left(\hat{a}_{ij}^{(r,t)}\right) \in \mathbb{R}^{s \times s}$ von A und $\left(\hat{b}_{jk}^{(t,u)}\right) \in \mathbb{R}^{s \times s}$ von B für $t = 1, \dots, p = \frac{m}{s}$ (Zeile 24):

$$\hat{c}_{ik}^{(r,u)} = \sum_{t=1}^{p} \sum_{j=1}^{s} \hat{a}_{ij}^{(r,t)} \cdot \hat{b}_{jk}^{(t,u)}.$$

Dabei werden für jeden Iterationsschritt t zuerst die Einträge der Submatrizen von A und B aus dem globalen Speicher in den gemeinsam genutzten Speicher geladen, wobei jeder Thread jeweils ein Element pro Submatrix lädt (Zeilen 34 & 35). Anschließend werden die Threads synchronisiert, um sicherzustellen, dass alle Daten geladen sind, bevor diese weiterverarbeitet werden. Jedes Mal, bevor der Block vorwärts schreitet, werden die Threads erneut synchronisiert.
Nachdem über alle Submatrizen von A und B iteriert wurde, werden die Elemente der Submatrix von C jeweils von dem Thread, der das Element berechnet hat, in den globalen Speicher geschrieben (Zeile 48). Jedes Element a_{ij} wird somit zwar noch n-mal aus dem Shared Memory aufgerufen aber nur noch q-mal aus dem globalen Speicher. Auf diese Weise können die Speicherzugriffe nicht nur parallelisiert, sondern auch die sehr langsamen Zugriffe auf den globalen Speicher deutlich reduziert werden.

Programm 4: CUDA-*Kernel* zur blockweisen Multiplikation zweier Matrizen A und B.

```
1  __global__ void Matrixmultiplikation(float *A, float *B, float *C, int wA
       ) {
2      const uint BLOCK_SIZE = blockDim.x;
3      const uint wB = gridDim.x * BLOCK_SIZE;
4
5      const uint bx = blockIdx.x;
6      const uint by = blockIdx.y;
7
8      const uint tx = threadIdx.x;
9      const uint ty = threadIdx.y;
10
11     /* Start- und Endposition in A */
12     const uint aBegin = wA * BLOCK_SIZE * by;
13     const uint aEnd = aBegin + wA;
14
15     /* Startposition in B */
16     const uint bBegin = BLOCK_SIZE * bx;
17
18     /* Schrittweite der Bloecke */
19     const uint aStep = BLOCK_SIZE;
20     const uint bStep = BLOCK_SIZE * wB;
21
22     /* Iteration ueber alle Bloecke */
23     float Csub = 0;
24     for (int a = aBegin, b = bBegin;
25          a < aEnd;
26          a += aStep, b += bStep)
27          {
28          /* Reservierung des Speicherplatzes im Shared Memory */
29          __shared__ float Asub[BLOCK_SIZE][BLOCK_SIZE];
30          __shared__ float Bsub[BLOCK_SIZE][BLOCK_SIZE];
31
32          /* Jeder Thread laedt ein Element */
33          /* vom Global in den Shared Memory */
34          Asub[ty][tx] = A[a + wA * ty + tx];
35          Bsub[ty][tx] = B[b + wB * ty + tx];
36          __syncthreads();
37
38          /* Multiplikation der Submatrizen */
39          for (int k = 0; k < BLOCK_SIZE; ++k)
40              {
41              Csub += Asub[ty][k] * Bsub[k][tx];
42              }
43          __syncthreads();
44          }
45
46     /* Jeder Thread schreibt einen Matrizenwert */
47     const uint c = wB * BLOCK_SIZE * by + BLOCK_SIZE * bx;
48     C[c + wB * ty + tx] = Csub;
49     }
```

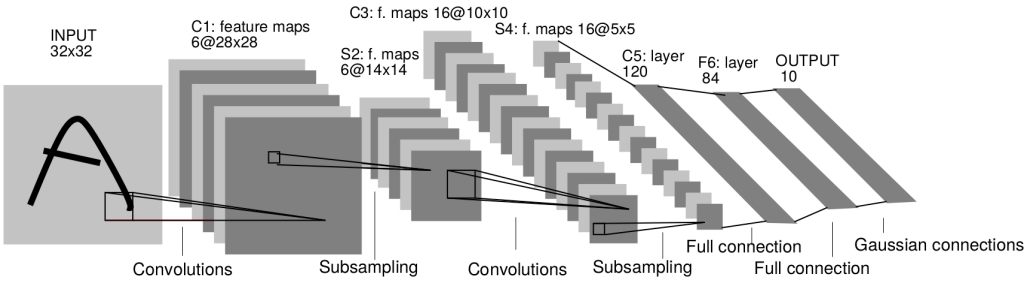

Abbildung 29: Architektur des LeNet-5 *Convolutional Neural Network*. Das Netzwerk besteht aus zwei *Convolutional* Layern gefolgt von je einem *Subsampling* Layer. Den Abschluss bilden drei vollverbundene Schichten [Lecun et al., 1998].

4.3 Deep Convolutional Neural Networks

Der Erfolg künstlicher neuronaler Netze ist unter anderem auf den technologischen Fortschritt bei parallelen Computerarchitekturen wie GPUs zurückzuführen, die die für das Training der Netze (Abschnitt 4.2.1) notwendigen Matrixmultiplikationen äußerst effizient berechnen können (Abschnitt 4.2.2). Zur Verarbeitung mit einem *Fully Connected Neural Network* (FCNN) können 2D- oder 3D-Bilddaten zeilenweise eingelesen und jedes Pixel einem Eingabeneuron zugewiesen werden. In der Bild-, aber auch Text- und Sprachverarbeitung haben sich jedoch sogenannte *Convolutional Neural Networks* (CNNs), die die lokale Struktur der Daten berücksichtigen, als äußerst erfolgreich erwiesen. Bei einem 2D-Bild beispielsweise durchläuft dabei ein sogenannter Filter das Bild von links nach rechts und von oben nach unten und berechnet dabei die Werte der nächsten Schicht (Abb. 29 & 30). Die Werte ergeben sich jeweils durch die Anwendung einer Aktivierungsfunktion auf die gewichtete Summe der Eingangsdaten und gegebenenfalls der Addition eines sogenannten *Bias* (Abb. 30 links). Insbesondere werden dabei auf jeden Ausschnitt des Bildes die gleichen Gewichte angewandt, wohingegen bei einem FCNN jedem Pixel der Eingangsschicht oder jedem Neuron in einer verdeckten Schicht genau ein Gewicht zugeordnet wird. Die Anzahl der je Schicht verwendeten Filter kann individuell gewählt werden. In der klassischen Netzwerkarchitektur *LeNet-5* werden in der ersten verdeckten Schicht 6 Filter und in der zweiten verdeckten Schicht 16 Filter verwendet (Abb. 29). Jeder Filter erzeugt einen der sogenannten *Channel* der darauffolgenden Schicht (Abb. 30 rechts). Für die Verarbeitung eines 2D-Bildes kann ein CNN formal wie folgt definiert werden:

Definition 4.20. *Für ein Bild x mit $m \times n$ Pixeln und Kernelmatrix*

$$K = \begin{pmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{pmatrix} \in \mathbb{R}^{3\times 3},$$

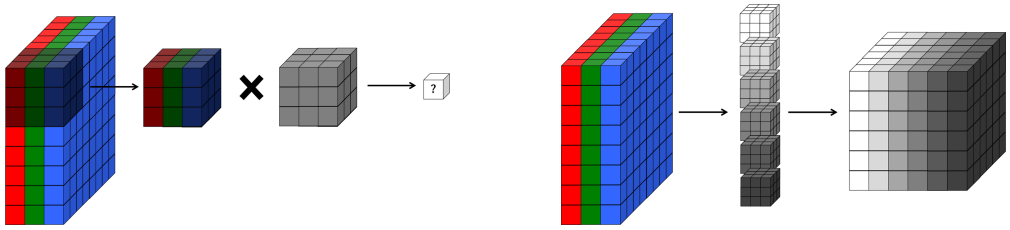

Abbildung 30: Convolution eines 2D-Bildes mit 3 Channeln durch einen 3×3-Filter. Die Anwendung des Filters (grau) auf einen Bildausschnitt liefert einen Wert der nächsten Schicht (**links**). Die Anwendung mehrerer Filter (verschiedene Graustufen) auf alle Teilbereiche des Bildes liefert für jeden Filter einen Channel der nächsten Schicht (**rechts**) [Buduma and Locascio, 2017].

sei die Convolution Matrix $W \in \mathbb{R}^{(mn)\times(m+2)(n+2)}$

$$\begin{pmatrix} w_1 & w_2 & w_3 & 0 & \cdots & 0 & w_4 & w_5 & w_6 & 0 & \cdots & 0 & w_7 & w_8 & w_9 & 0 & \cdots & \cdots & 0 \\ 0 & w_1 & w_2 & w_3 & 0 & \cdots & 0 & w_4 & w_5 & w_6 & 0 & \cdots & 0 & w_7 & w_8 & w_9 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \cdots & \ddots & \ddots & \ddots & \ddots & \ddots & \cdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \cdots & \ddots & \ddots & \ddots & \ddots & \ddots & \cdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & w_1 & w_2 & w_3 & 0 & \cdots & 0 & w_4 & w_5 & w_6 & 0 & \cdots & 0 & w_7 & w_8 & w_9 \end{pmatrix}$$

gegeben, wobei der Rand des Bildes mit Nullen erweitert wird. Daraus ergibt sich für einen Convolutional Layer mit C *Kanälen und* F *Filtern die Operatormatrix*

$$T = \begin{pmatrix} W_{1,1} & W_{1,2} & \cdots & W_{1,C} \\ W_{2,1} & W_{2,2} & \cdots & W_{2,C} \\ \vdots & \vdots & \ddots & \vdots \\ W_{F,1} & W_{F,2} & \cdots & W_{F,C} \end{pmatrix} \in \mathbb{R}^{Fmn \times C(m+2)(n+2)},$$

wobei jedes Element eine dünnbesetzte Convolution Matrix mit 9 Parametern ist. Ein Deep Convolutional Neural Network der Tiefe $J \in \mathbb{N}$ *ist eine Sequenz von Vektoren*

$$h^{(j)} = \sigma\left(T^{(j)}\hat{h}^{(j-1)} + b^{(j)}\right), \qquad j = 1, \ldots, J,$$

wobei $C^{(j)} = F^{(j-1)}$ *für* $j \geq 2$, $\hat{h}^{(j)} \in \mathbb{R}^{F^{(j)}(m+2)(n+2)}$ *entspricht der mit Nullen aufgefüllten Ausgabe* $h^{(j)} \in \mathbb{R}^{F^{(j)}mn}$ *und* $\hat{h}^{(0)} = \hat{x} \in \mathbb{R}^{C^{(1)}(m+2)(n+2)}$ *das erweiterten Bild mit* $C^{(1)}$ *Kanälen. Dabei wird die Aktivierungsfunktion* σ *komponentenweise angewandt und für die Elemente der Bias Vektoren* $b^{(j)} \in \mathbb{R}^{F^{(j)}mn}$ *gilt:*

$$\begin{aligned} b_1^{(j)} &= \cdots = b_{mn}^{(j)}, \\ b_{mn+1}^{(j)} &= \cdots = b_{2mn}^{(j)}, \\ &\vdots \qquad\qquad \vdots \\ b_{(F-1)mn+1}^{(j)} &= \cdots = b_{Fmn}^{(j)}. \end{aligned}$$

Beispiel 4.21. Das CNN *LeNet-5* (Abb. 29) zur Klassifizierung handgeschriebener Zahlen von 0-9 der Größe 32×32 Pixel benötigt insgesamt 61.470 freie Parameter (ohne Berücksichtigung der *Bias*-Parameter). Für die *Convolutional* Layer wird eine Kernelgröße von 5×5 mit einer Schrittweite von einem Pixel und für die *Subsampling* Layer eine Kernelgröße von 2×2 mit einer Schrittweite von 2 Pixel angenommen. Der erste *Convolutional* Layer verfügt über 6, der zweite über 16 Filter. Da jeder Filter seinen eigenen Kernel besitzt, ergeben sich für den ersten *Convolutional* Layer $6 \times 5 \times 5$ Parameter. Durch die Faltung reduziert sich die „Bildgröße“ auf 28×28 Pixel. Der Nachfolgende *Subsampling* Layer wählt für jeden 2×2 Pixel großen Ausschnitt den jeweils größten Wert. Ohne Auffüllen der Ränder mit Nullen reduziert sich dadurch die „Bildgröße“ auf 14×14 Pixel. Der folgende *Convolutional* Layer hat $6 \times 16 \times 5 \times 5$ freie Parameter, da er über 6 Eingangschannel und 16 Filter verfügt. Durch die Faltung und den nachgelagerten *Subsampling* Layer wird die „Bildgröße“ auf 5×5 Pixel reduziert. Den Abschluss bilden zwei vollverbundene, verdeckte Schichten und eine Ausgabeschicht. Hierfür werden 400×120, 120×84 und 84×10 weitere Parameter benötigt. Insgesamt ergeben sich somit 61.470 Parameter.

CNNs sind insbesondere dann effizient, wenn es um die Approximation hochdimensionaler Daten mit einem großen Anteil an redundanten Informationen und lokalen Strukturen geht. Wie dies zum Beispiel bei Bildern der Fall ist. Da die Filter über alle möglichen 3×3 Bildausschnitte iterieren, ist das Netzwerk gezwungen, hochgradig verallgemeinerbare Informationen zu lernen. Illustrativ könnte ein Filter für jedes Sichtfeld die Frage beantworten, ob eine bestimmte Struktur, wie zum Beispiel eine Kante, enthalten ist oder nicht. Durch Wiederverwendung der Gewichte wird die Anzahl der freien Parameter im Vergleich zu einem FCNN erheblich reduziert. Aus Definition 4.20 & Beispiel 4.21 geht hervor, dass das CNN *LeNet-5* (Abb. 29) insgesamt 61.470 freie Parameter besitzt, wohingegen ein klassisches künstliches neuronales Netz mit einer verdeckten Schicht bestehend aus 500 Neuronen für die gleiche Aufgabe, d. h. die Klassifizierung von 10 handgeschriebenen Zahlen der Größe 32×32 Pixel, bereits 517.000 freie Parameter benötigt ($32 \cdot 32 \cdot 500 + 500 \cdot 10 = 517.000$). Der Einfachheit halber wurden die *Bias*-Parameter hier nicht berücksichtigt.
Durch die Reduktion der Gewichte wird die Verarbeitung redundanter Informationen vermieden. Während ein FCNN viel Aufwand erfordert, um die wesentlichen Informationen aus den Bilddaten zu extrahieren, wird bei einem CNN die Informationsmenge durch das Modell selbst auf die Verarbeitung lokaler Muster beschränkt. Die Reduzierung der Gewichte verringert zudem das Risiko einer Überanpassung auf die Trainingsdaten (*Overfitting*). Hierbei passt sich das Modell zu stark oder gar exakt auf die Trainingsdaten an, ist gleichzeitig aber nicht in der Lage unbekannte Daten richtig vorherzusagen. Der Grund hierfür ist, dass das Netzwerk gelernt hat, bestimmte Muster in den Trainingsdaten zu erkennen, die jedoch nicht verallgemeinert werden können.
Im Allgemeinen muss es eine Balance zwischen der Netzwerkgröße (d. h. der Anzahl freier Parameter) und dem in den Trainingsdaten vorhandenen Informationsgehalt geben. Ist das Netzwerk zu groß gewählt, beginnt das Netz spezifische Eigenschaften der Trainingsdaten zu lernen, die nicht auf andere Daten übertragbar sind. Ist das Netzwerk hingegen zu klein gewählt, reichen die Gewichte nicht aus, um genügend Informationen abspeichern zu können. Um Verallgemeinerungen treffen und gleichzeitig *Overfitting* vermeiden zu können,

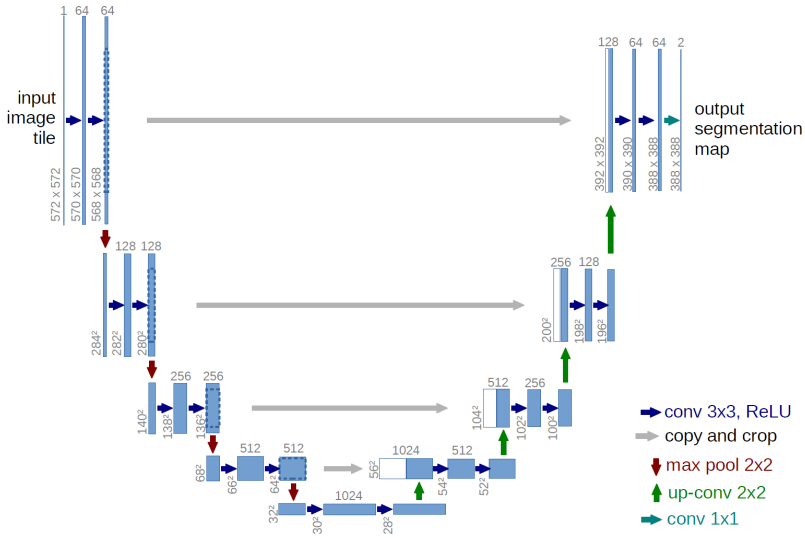

Abbildung 31: Netzwerkarchitektur eines *Convolutional Neural Network* mit zweidimensionaler Eingabe- und Ausgabeschicht sowie U-Net Struktur [Ronneberger et al., 2015]. Nach einem Block von zwei aufeinanderfolgenden 3×3 *Convolutional* Schichten folgt eine *Maxpooling* bzw. eine *Upconvolutional* Schicht. Die Zahl über den Schichtsymbolen entspricht der Anzahl der jeweiligen Channel. Am Ende erfolgt eine 1×1 *Convolutional* Schicht, wobei die Anzahl der Filter der Anzahl der Label entspricht.

werden häufig enorme Datenmengen benötigt. Die Architektur von CNNs ermöglicht eine hoch parallele Vorgehensweise bei der Optimierung der Gewichte (Abschnitt 4.2.2) und somit einen massiven Durchsatz von Trainingsdaten.

4.3.1 Netzwerkarchitektur & Optimierung

Zur automatischen Segmentierung von 3D-Bilddaten wird in dieser Arbeit ein 3D-CNN konstruiert, das der typischen Struktur eines U-Nets [Ronneberger et al., 2015] folgt. Da das gesamte Volumen in der Regel zu groß ist, um verarbeitet zu werden, dienen $64 \times 64 \times 64$ große Ausschnitte des volumetrischen Bildes als Eingangsdaten. Die Bilddaten müssen vor der Verarbeitung entsprechend zerlegt und anschließend wieder zu einem Gesamtbild zusammengefügt werden. Die Architektur des Netzes besteht aus einem kontrahierenden (Encoder) und einem expandierenden (Decoder) Teil (Abb. 31). In beiden Teilen erfolgt eine sich wiederholende Anwendung von zwei aufeinander folgenden $3 \times 3 \times 3$ *Convolutional* Layern, wobei auf jede Faltung je eine Batch-Normalisierung und eine ReLU-Aktivierungsschicht (Abschnitt 4.1.1) folgt. Auf jeden dieser Blöcke folgt im Encoder-Teil eine $2 \times 2 \times 2$ *Maxpooling* Schicht mit einer Schrittweite von 2 Pixel, um die Bilddimension sukzessive zu reduzieren. Nach jedem dieser Reduktionsschritte wird die Anzahl der Filter, ausgehend von 32 Filtern, verdoppelt. Im ursprünglichen U-Net (Abbildung 31) wird mit 64 Filter gestartet. Das hier verwendete Netzwerk erreicht dabei, wie auch das ursprüngliche

2D U-Net, maximal 1024 Channel. Jeder Block im Decoder-Teil startet mit einer Verkettung der Upconvolution der vorherigen Schicht mit der letzten Schicht des korrespondierenden Blocks im Encoder-Teil, gefolgt von zwei $3 \times 3 \times 3$ *Convolutional* Layern, wiederum jeweils gefolgt von einer Batch-Normalisierung und einer ReLU-Aktivierungsschicht. In der letzten Schicht wird ein $1 \times 1 \times 1$ *Convolutional* Layer verwendet, um jeden Feature Vektor (Einträge aller Channel an der gleichen Position) auf die gewünschte Anzahl an Klassen zu projizieren.
Das Training des Netzwerks (d. h. die Optimierung der Gewichte) erfolgt mittels *Stochastic Gradient Descent* (Abschnitt 4.2.1) und einer Lernrate von 0.01, *Decay* von 10^{-6}, aktiviertem Nesterov Momentum von 0.9, insgesamt 200 Trainingsepochen, und einer Batchgröße von 24. Alle Trainingsdaten werden zuvor auf das Intervall 0 bis 1, denselben Mittelwert und die gleiche Varianz skaliert.

4.4 Segmentierung von Honigbienengehirnen

Alle Tiere, von den Insekten bis zu den Menschen, weisen eine große Vielfalt an Verhaltensweisen auf, die von einer kognitiven Variabilität untermauert wird [Dunbar, 1998]. Die kognitive Variabilität ist die Grundlage für Anpassungen an Umweltveränderungen. Ein tieferes Verständnis der natürlichen Variation von Gehirngrößen, -architekturen und kognitiven Leistungen ist daher von großer Bedeutung. Untersuchungen zur Gehirngröße und -organisation beschränken sich jedoch aufgrund des manuellen Aufwands meist auf wenige Stichproben. Eine geringe Stichprobengröße macht es jedoch schwierig, einen statistischen Zusammenhang zwischen morphologischen Variationen des Gehirns und kognitiver Variation herzustellen. Nur die Analyse einer großen Anzahl von Gehirnen ist dazu in der Lage, geringfügige jedoch statistisch und biologisch relevante Variationen der Gehirngrößen und -architekturen aufzudecken.
Das in Abschnitt 4.3.1 entwickelte CNN ermöglicht eine automatisierte Segmentierung großer Stichprobenserien und somit eine systematische vergleichende quantitative Analyse, die ökologische und evolutionsbiologische Fragen der Neurowissenschaften beantworten kann: Sind größere Gehirne leistungsfähiger [Luders et al., 2009, Pietschnig et al., 2015]? Welchen Einfluss haben soziale und ökologische Faktoren auf die Gehirnevolution [Dunbar, 1998]? Wie wirken sich Umweltstressoren auf die Entwicklung und Kognition des Gehirns aus [Smith et al., 2020]?
Honigbienen sind mit Gehirnen ausgestattet, die kleiner als 1 mm^3 sind. Nichtsdestotrotz zeigen sie eine ausgeprägte interindividuelle Verhaltensvariabilität innerhalb und zwischen Kolonien [Jandt et al., 2014], was sie zu idealen Organismen macht, um die kognitiven Funktionen von Insekten und die zugrunde liegenden neuronalen Substrate zu untersuchen [Giurfa, 2013, Menzel, 2012]. Die Verhaltensvariabilität ist von zentraler Bedeutung für die Arbeitsteilung und die Koloniefunktion. Ein Teil dieser Variabilität wurde mit der Reifung bestimmter Hirnareale aufgrund einer zunehmenden Spezialisierung und der Erfahrung bei der Nahrungssuche in Verbindung gebracht [Fahrbach et al., 1998, Withers et al., 1993]. Es besteht jedoch noch kein eindeutiger Zusammenhang zwischen der Neuroarchitek-

tur der Bienen und ihrer Verhaltensvarianz, da Daten aus vergleichenden Analysen fehlen. Die meisten Studien haben sich bisher auf kleine Stichprobengrößen (ungefähr 10 Einzeldaten) konzentriert. Studien auf der Bevölkerungsebene sind hingegen selten [Sombke et al., 2015].

4.4.1 Automatische Segmentierung von 110 Honigbienengehirnen

Als Grundlage für eine statistische Analyse [Lösel et al., Vorb] dienen μCT-Scans von 120 Honigbienen-Sammlerinnen (*Apis mellifera*, Buckfast) aus zwei Bienenpopulationen (Population A: 100 Bienen aus 6 Bienenstöcken, Population B: 20 Bienen aus 3 Bienenstöcken), die aus der Umgebung von Toulouse (Frankreich, August 2020) stammen. Nach der Präparation der Bienenköpfe [Smith et al., 2016] wurden diese mit einer Auflösung von 5,4 μm isotroper Voxelgröße gescannt. Von den 120 μCT-gescannten Gehirnen wurden 10 während der Präparation beschädigt und für die weitere Analyse verworfen. Insgesamt verblieben somit 110 Gehirne von Honigbienen. Die Bildgrößen und -auflösungen variieren je nach 3D-Volumen und betragen im Durchschnitt $844 \times 726 \times 485$ Voxel bzw. $0,0054 \times 0,0054 \times 0,0054$ mm^3. Vor der Verarbeitung wurde jeder Datensatz mit Avizo 2019.1 (Thermo Fisher Scientific, Waltham, USA) manuell auf den Bereich der Neuropile (Abb. 32d) zugeschnitten, was zu einer durchschnittlichen Größe von $451 \times 273 \times 167$ Voxel führte.
Basierend auf dem 3D-Bienengehirnatlas [Brandt et al., 2005] wurden sechs große Neuropile (ohne Zellkörper) analysiert: die Antennalloben (AL), die Mushroom Bodies (MB) (bestehend aus medialen und lateralen Kelch, Stiel und Lappen), der zentrale Komplex (CX) (bestehend aus dem Zentralkörper, den paarigen Knoten und der protozerebralen Brücke), den Medullae (ME) und Lobulae (LO) (kombiniert als optischer Lappen (OL), die Retinae und Laminae wurden nicht gemessen) und andere Neuropile (OTH, Protocerebrallappen und subösophageales Ganglion) (Abb. 32).
Die volumetrische Analyse der Neuropile erfordert die Isolierung der verschiedenen Regionen aus den CT-Scans durch Segmentierung. Da sich die Strukturen in allen Gehirnen wiederholen und eine manuelle oder semi-automatische Segmentierung für so viele Exemplare einen erheblichen Aufwand bedeutet, wurde das in Abschnitt 4.3.1 beschriebene künstliche neuronale Netz für eine automatisierte Segmentierung der Daten verwendet. Hierfür wurden zunächst 26 Datensätze, jeweils bestehend aus dem CT-Scan und einer bereits vollständigen Rekonstruktion der darin enthaltenen Neuropile verwendet (Abschnitt 5.1), um das künstliche neuronale Netz zu trainieren. Das Training des Netzes dauerte unter Verwendung von 4 NVIDIA Tesla V100 etwa 13 Stunden. Anschließend wurde das trainierte Netz verwendet, um die übrigen 84 μCT-Scans automatisch zu segmentieren. Die automatische Segmentierung der Daten dauerte mit einer NVIDIA GeForce GTX 1080 Ti durchschnittlich 21 Sekunden.

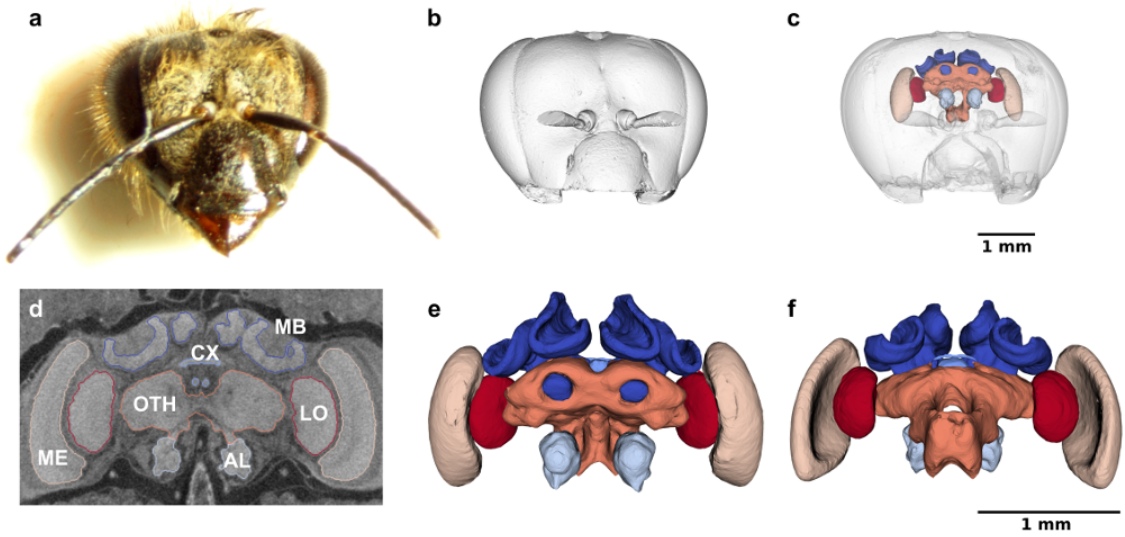

Abbildung 32: Automatische Segmentierung von Honigbienengehirnen.
a Frontalansicht des Kopfes einer Sammelbiene. **b** Oberflächenrendering des Kopfes mit entfernten Mandibeln. **c** Überlagerung des Kopfes und der rekonstruierten Neuropile. **d** Frontaler Querschnitt des Tomogramms mit den Segmentierungsgrenzen der Mushroom Bodies (MB), des zentralen Komplexes (CX), der Antennalloben (AL), der Medullae (ME), der Lobulae (LO) und anderer Neuropile (OTH). **e** Frontalansicht der rekonstruierten MB (dunkelblau), CX (himmelblau), AL (hell-himmelblau), ME (beige), LO (rot) und OTH (orange). **f** Dorsalansicht der rekonstruierten Neuropile [Lösel et al., Vorb].

4.4.2 Evaluation der automatischen Segmentierung

Um die Genauigkeit der Segmentierungsergebnisse zu bewerten, wurde, wie zuvor bei der Evaluation der GPU-basierten Random Walks (Abschnitt 3.7), der Dice-Ähnlichkeitskoeffizient (Dice) verwendet. Der Dice-Score quantifiziert die Übereinstimmung zweier Segmentierungen und liegt zwischen 0 und 1, wobei 0 keine Überlappung und 1 eine perfekte Übereinstimmung der beiden Segmentierungen bedeutet. Für die Evaluation wurde nur die *Cleanup*-Funktion von Biomedisa (Abschnitt 5.2.2) verwendet, um Ausreißer und freistehende Inseln automatisch zu entfernen. Für die Auswertung der automatischen Segmentierung wurden die Bilddaten in zwei Teilmengen unterteilt. Zum einen die 26 Trainingsbilder und zum anderen die 84 verbliebenen Testbilder. Die Evaluation des trainierten Netzes erfolgte anhand der automatisch erstellten Segmentierungen der 84 Testbilder. Hierfür wurden alle 84 Ergebnisse von einem/einer biologischen Experten/-in manuell validiert und gegebenenfalls korrigiert. Anschließend wurde das ursprüngliche Segmentierungsergebnis mit der manuell korrigierten Version verglichen, um die Genauigkeit der automatisierten Segmentierung und die Größe des Fehlers zu bestimmen.
Insgesamt wurde auf diese Weise ein Dice-Score von 0,988 erreicht (Tabelle 3). Bei 10,7 % der 84 Testbilder erforderte das Ergebnis keine oder nur eine geringe manuelle Korrektur. Dies war immer dann der Fall, wenn der Fehler kleiner als 0,01 % betrug (Abb. 33a).

Bei 84,5 % war eine leichte manuelle Korrektur erforderlich, die etwa 1 bis 2 Minuten beanspruchte. Hier lag der Fehler zwischen 0,01 % und 4 % (Abb. 33b). Nur 4,8 % der Segmentierungsergebnisse waren signifikant fehlerbehaftet, wobei der Fehler größer als 4 % betrug (Abb. 33c). Dies wurde in der Regel durch eine starke Abweichung der Bilddaten von den Trainingsdaten (z. B. aufgrund einer unvollständigen Färbung oder einer Beschädigung bei der Probenvorbereitung) verursacht und machte eine umfangreiche manuelle Korrektur oder sogar eine vollständige halbautomatische Rekonstruktion durch die GPU-basierten Random Walks (Kapitel 3) erforderlich.
Um die Segmentierungsgenauigkeit bei einer zunehmenden Anzahl von Trainingsbildern zu bestimmen, wurde das Netz basierend auf $3, 7, 12, 18$ und 26 Bildern erneut trainiert und jeweils die Genauigkeit auf den 84 Testbildern gemessen (Tabelle 3). Für jede Anzahl wurden dabei mehrmals verschiedene Trainingsbilder ausgewählt und anschließend der Durchschnitt der Genauigkeiten berechnet. Zum Beispiel wurden die 26 Trainingsbilder in 9 Teilmengen mit je 3 Trainingsbildern unterteilt, wobei ein Bild zweimal verwendet wurde. Da die 84 Testdaten manuell korrigierte Versionen des ursprünglichen Netzes sind, wurde das Netz auch auf allen 26 Trainingsbildern erneut trainiert, um die sonst entstehende Verzerrung aufzulösen. Des Weiteren wurde das Modell auf den Gesamtvolumen getestet. Hierbei zeigte sich, dass das vorherige Zuschneiden der Bilddaten auf den Bereich der Neuropile im Vergleich zu den unbeschnittenen Daten deutlich bessere Ergebnisse liefert (Tabelle 3).
Insgesamt ermöglichte das künstliche neuronale Netz eine hochpräzise und schnelle Segmentierung der 84 Testbilder. Die Segmentierungsergebnisse erforderten lediglich eine manuelle Korrektur von 1,2 % (Tabelle 3). Größere Korrekturen waren meist nur für CX (insgesamt 3,3 %) erforderlich. Die feine Struktur des Neuropils in Kombination mit einem niedrigen Kontrast im Tomogramm machte es oft, selbst mit bloßem Auge, schwierig das Neuropil im CT-Scan zu erkennen. Der Einsatz des künstlichen neuronalen Netzes lieferte somit qualitativ hochwertige Ergebnisse, bei einem gleichzeitig erheblich reduzierten Arbeitsaufwand gegenüber herkömmlichen Methoden. Bei einer ausschließlich manuellen oder halbautomatischen Segmentierung hätte die Segmentierung für jeden Bildstapel deutlich länger gedauert. Die rein manuelle Segmentierung (*Slice-by-Slice*) benötigt etwa 5 bis 6 Stunden pro Gehirn. Eine Vorsegmentierung jeder fünften Schicht und der Einsatz der GPU-basierten Random Walks reduziert den manuellen Aufwand auf immer noch 2-3 Stunden.

4.4.3 Künstliche Erweiterung der Trainingsdaten

Neben der stark vorangeschrittenen Entwicklung spezifischer Computerhardware zur Berechnung des Optimierungsprozesses (Abschnitt 4.2.1 & 4.2.2), liegt ein weiterer wesentlicher Grund des jüngsten Erfolgs künstlicher neuronaler Netze in der Generierung, Aufbereitung und Bereitstellung großer Mengen von Trainingsdaten. Während beim klassischen maschinellen Lernen dem Algorithmus die gesuchten Muster (sogenannte *Features*) explizit mitgeteilt werden müssen (z. B. welcher Grauwertbereich oder welche Form einem bestimmten Objekt zugeordnet werden kann), sind künstliche neuronale Netze dazu in der Lage,

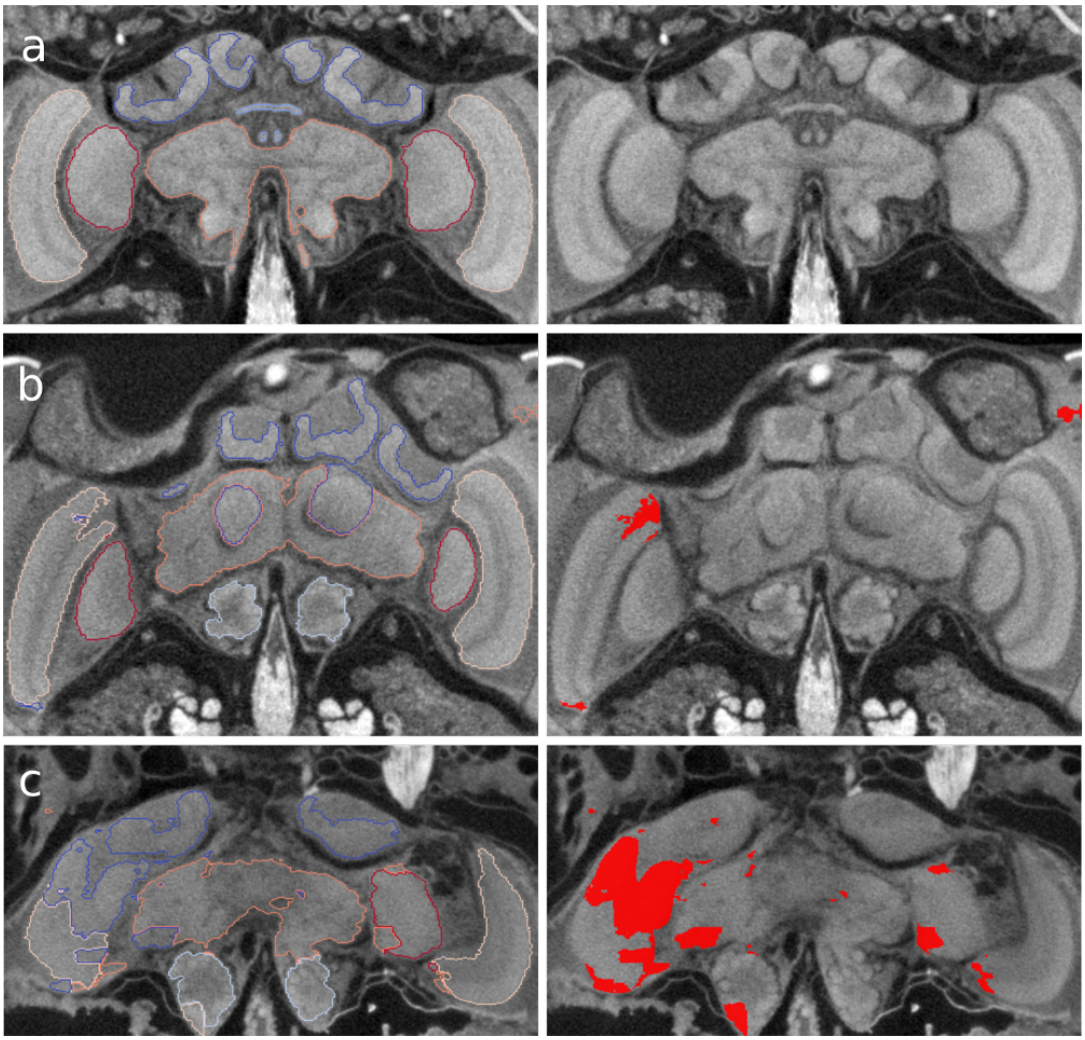

Abbildung 33: Segmentierungsergebnisse (links) und Segmentierungsfehler (rechts) eines Deep Neural Network trainiert auf Grundlage von 26 semi-automatisch erstellten Trainingsdaten. a Korrekte Segmentierung ohne Fehler. **b** Teilweise fehlerhaftes Segmentierungsergebnis mit einem typischen Ausreißer auf der rechten Bildseite (Genauigkeit: ME 97,6 %, total 99,1 %). **c** Stark fehlerbehaftete Segmentierung (Genauigkeit insgesamt 87,7 %) [Lösel et al., Vorb].

Tabelle 3: Durchschnittliche Dice Scores der semi-automatischen und automatischen Segmentierung. Als Referenz dienen die manuell korrigierten Ergebnisse. Ausreißer wurden vor der Evaluation automatisch entfernt. Die letzte Spalte entspricht der prozentual benötigten Korrektur.

Datensatz/Methode	AL	MB	ME	LO	CX	OTH	Total	Error
84 Testdaten	0,984	0,982	0,992	0,991	0,967	0,988	**0,988**	**1,2 %**
3 Trainingsdaten	0,878	0,870	0,947	0,938	0,601	0,894	**0,905**	**9,5 %**
7 Trainingsdaten	0,931	0,920	0,971	0,966	0,822	0,940	**0,945**	**5,5 %**
12 Trainingsdaten	0,955	0,938	0,978	0,980	0,878	0,954	**0,959**	**4,1 %**
18 Trainingsdaten	0,957	0,944	0,981	0,981	0,896	0,961	**0,964**	**3,6 %**
26 Trainingsdaten	0,967	0,955	0,985	0,983	0,911	0,968	**0,971**	**2,9 %**
Gesamtvolumen	0,918	0,894	0,956	0,947	0,752	0,924	**0,926**	**7,4 %**
Random Walks	0,967	0,949	0,986	0,982	0,856	0,962	**0,967**	**3,3 %**
Avizo Interpolation	0,925	0,925	0,915	0,914	0,848	0,946	**0,928**	**7,2 %**

die für eine akkurate Klassifizierung oder Segmentierung notwendigen Bildeigenschaften aus den Trainingsdaten selbst zu extrahieren.
Bei CNNs geht es dabei vor allem darum, lokale Eigenschaften mit den zu segmentierenden Objekten in Verbindung zu bringen. Während die erste Schicht noch die tatsächlichen Bildeigenschaften, wie Kanten identifiziert, werden in den darauffolgenden Schichten abstrakte Bilddaten verarbeitet, die sich aus den vorhergehenden Schichten ergebenen. CNNs können dabei, aus der Kombination solcher *Features*, Rückschlüsse auf die Eingangsdaten schließen. Durch sehr tiefe künstliche neuronale Netze lassen sich sehr viele *Features* berücksichtigen, sehr viel mehr als bei herkömmlichen Verfahren. Darüber hinaus ist es nicht notwendig, all diese *Features* explizit zu implementieren.
Ein unerwünschter Nebeneffekt tiefer Netze ist jedoch das sogenannte *Overfitting* auf die Trainingsdaten. Das künstliche neuronale Netz ist dabei in der Lage, so viele Informationen abzuspeichern, dass die einzelnen Bilder auf Basis individueller Bildinformationen, wiedererkannt werden können. Die Folge ist einerseits eine sehr gute Vorhersagbarkeit der Trainingsdaten, andererseits werden jedoch keine generischen, das heißt allgemeingültigen, Bildeigenschaften erlernt, so dass die Genauigkeit bei unbekannten Daten in der Regel deutlich schlechter ausfällt. Der Schlüssel für ein erfolgreiches Training liegt folglich in der Balance zwischen Netzwerkgröße (d. h. Informationsspeicherkapazität) und der in den Trainingsdaten enthaltenen Informationsmenge.
Da die Erstellung von Trainingsdaten oft sehr mühsam ist und somit nicht immer genügend Trainingsdaten zur Verfügung stehen, werden die Trainingsdaten häufig künstlich erweitert. Hierfür eignet sich neben einfachen Methoden, wie Rotation, Translation und Spiegelung, eine elastische Deformation des ursprünglichen Bildes, um einen leicht abweichenden Trainingsdatensatz zu erzeugen, der sich jedoch immer noch im Rahmen der natürlichen Erscheinung, bspw. der Bienengehirne, befindet (Fig. 34). Auf diese Weise wird im Optimierungsprozess eine Probe niemals in identischer Weise wiederverwendet. Dies wirkt einem *Overfitting* auf individuelle Merkmale der Bilder entgegen.

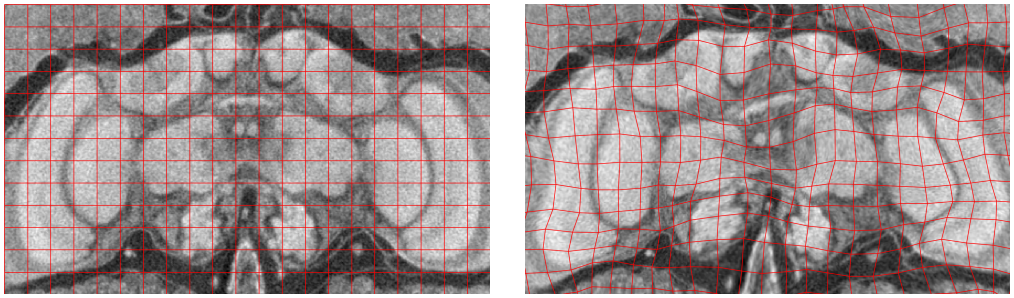

Abbildung 34: Auswirkung einer elastischen Deformation auf einen μCT-Scan eines Honigbienengehirns. links CT-Scan ohne Deformation. **rechts** CT-Scan nach elastischer Deformation.

Seien $\mathcal{I}$ die Bilddaten der Größe $l \times m \times n$ Voxel, *Rand* ein Array von gleicher Größe mit gleichverteilten Zufallszahlen auf dem Intervall $[-1, 1]$, G ein Gaußfilter, α die maximal mögliche Verschiebung eines Voxels und σ die Stärke der Glättung des Gaußfilters, dann erzeugt die Transformation

$$\mathcal{I}_t(i + \Delta_x(i,j,k), j + \Delta_y(i,j,k), k + \Delta_z(i,j,k)) = \mathcal{I}(i,j,k),$$

mit

$$\begin{aligned} \Delta_x &= G_\sigma(\alpha Rand), \\ \Delta_y &= G_\sigma(\alpha Rand), \\ \Delta_z &= G_\sigma(\alpha Rand), \end{aligned}$$

für $i = 1, \ldots, l$, $j = 1, \ldots, m$, $k = 1, \ldots, n$ eine elastische Deformation $\mathcal{I}_t$ des ursprünglichen Bildes. Für die Honigbienengehirne wurde $\alpha = 2000$ und $\sigma = 20$ gewählt. Nach der Transformation der Voxel wurden diese mit einer Spline-Interpolation der Ordnung 1 und die Label mit Ordnung 0 (*Nearest Neighbour*) interpoliert. Auf den 84 Testdaten konnte die Genauigkeit der Segmentierungsergebnisse durch die elastische Deformation der Bilddaten während der Trainingsphase um 0,02 Prozentpunkte gesteigert werden. Interessant ist dabei vor allem die Tatsache, dass für den CT-Scan mit der zuvor niedrigsten Genauigkeit eine große Qualitätssteigerung erzielt werden konnte (*Head48*: von 0,795 auf 0,855). Der CT-Scan weist eine im Vergleich zu den Trainingsdaten starke Deformation auf, die möglicherweise durch eine äußere Manipulation (wie Quetschen) während der Probenvorbereitung verursacht wurde. Diese Veränderung wird jedoch nicht durch die Trainingsdaten abgebildet, wodurch das künstliche neuronale Netz nicht in der Lage ist, für diesen Datensatz eine adäquate Segmentierung zu ermöglichen. Durch die elastische Deformation der Trainingsdaten lernt das künstliche neuronale Netz jedoch, auch CT-Scans, die stark von den Trainingsdaten abweichen, genauer zu segmentieren.

5 Online-Segmentierungsplattform Biomedisa

Die hier vorliegende Dissertation entstand im Rahmen der interdisziplinären BMBF-Verbundprojekte ASTOR (**A**rthropoden-**S**trukturaufklärung mittels ultra-schneller **T**omographie und **O**nline **R**ekonstruktion) und NOVA (**N**etzwerk zur **O**nline-**V**isualisierung und synergistischen **A**nalyse von Tomographiedaten [Schmelzle et al., 2017]). Ziel dieser Projekte war es, neue Segmentierungsverfahren zu entwickeln und in eine Analyseinfrastuktur zu integrieren, mit der sich der Zeit- und Arbeitsaufwand für die Segmentierung der an den Synchrotrons ANKA des Karlsruher Instituts für Technologie (KIT) und Petra III am Deutschen Elektronen-Synchrotron (DESY) in Hamburg aufgenommenen Datensätze drastisch reduzieren und gleichzeitig die Qualität und Reproduzierbarkeit der Ergebnisse deutlich verbessern lässt, um dadurch eine effizientere Nutzung der wertvollen Strahlzeit an den tomographischen Synchrotronstrahllinien zu erzielen.
Die Synchrotron-Röntgen-Mikrotomographie (SR-μCT) bietet einzigartige Möglichkeiten für die morphologische Analyse von Tieren. Innere Strukturen sind auch in undurchsichtigen Organismen auf nicht-invasive, dreidimensionale Weise mit einer Auflösung von weniger als einem Mikrometer sichtbar. In den letzten Jahren konnte die räumliche und zeitliche Auflösung der SR-μCT stark erhöht und die Aufnahmezeit erheblich reduziert werden. Die Auswertung der Datensätze stieß jedoch, bedingt durch deren Größe und der Komplexität der darin abgebildeten Strukturen, längst an ihre Grenzen. Insbesondere die Segmentierung war mit den bisherigen Ansätzen nicht mehr zu bewerkstelligen.
In dieser Dissertation wurde deshalb ein neuartiges Verfahren zur halbautomatischen Segmentierung sehr großer volumetrischer Datensätze (Kapitel 3) sowie ein auf künstlichen neuronalen Netzwerken basierendes Verfahren zur automatischen Segmentierung vieler ähnlicher Proben (Kapitel 4) entwickelt. Diese Verfahren bilden die Kernfunktionen der benutzerfreundlichen Online-Anwendung Biomedisa (`https://biomedisa.org`), die in diesem Kapitel vorgestellt wird.
Die als Online-Plattform konzipierte Anwendung wurde bereits innerhalb der Projektlaufzeiten von ASTOR und NOVA für einige Studien erfolgreich eingesetzt. Eines der prominentesten Ergebnisse war dabei die Entdeckung und Beschreibung fossiler parasitärer Wespen, die in mineralisierten Fliegenpuppen eingeschlossen sind [van de Kamp et al., 2018]. Durch SR-μCT war es erstmals möglich, das Innere dieser Millionen Jahre alten Objekte zerstörungsfrei zu untersuchen und die darin enthaltenen inneren Strukturen dreidimensional zu rekonstruieren. In der Studie wurden über 1500 fossile Fliegenpuppen an der auf Hochdurchsatz spezialisierten Tomographie-Station UFO am KIT gescannt. Die resultierenden Datensätze hatten eine Größe von $2016 \times 2016 \times 2016$ Voxel. In 55 Fällen konnte ein parasitäres Ereignis beobachtet werden. Nachdem die parasitischen Wespen im Innern der Fliegenpuppen mithilfe von SR-μCT am KIT durchleuchtet worden waren, wurden sie unter Zuhilfenahme der Online-Anwendung Biomedisa hochauflösend rekonstruiert (Abb. 35 & 36). Die Studie lieferte den ersten direkten Nachweis eines Endoparasitismus, also den Nachweis eines Parasiten in seinem Wirt. Auf diese Weise konnten vier unbekannte

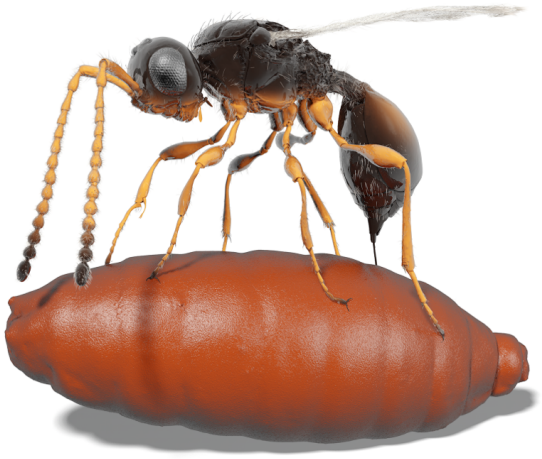

Abbildung 35: Digitale Auferstehung. Illustration der parasitischen Wespe *Xenomorphia resurrecta* (Abschnitt 3.5.7), die ein Ei in einer Fliegenpuppe ablegt. Die Grafik basiert auf der Rekonstruktion mit Biomedisa (Abb. 36 a-m) [van de Kamp et al., 2018].

Wespenarten entdeckt und erstmals beschrieben werden. Die neu entdeckten Arten lebten in einem Zeitraum von vor rund 40 bis 23 Millionen Jahren. Die am häufigsten beobachtete Art wurde *Xenomorphia resurrecta* genannt. Der Gattungsname „Xenomorphia“ erinnert dabei an das als Xenomorph bekannte Wesen aus der Science-Fiction-Filmreihe Alien, das lateinische Wort „resurrecta“ bedeutet auferstanden und bezieht sich auf die durch Biomedisa erzielte „digitale Auferstehung“ der Art (Abb. 35 & 36).
Biomedisa wird von den Projektpartnern/-innen und vielen anderen Partnern/-innen bereits routinemäßig zur Segmentierung großer volumetrischer Datensätze eingesetzt. Die Anzahl der auf der Online-Plattform durchgeführten Segmentierungen hat bereits während der Projektlaufzeiten stark zugenommen und der Nutzerkreis hat sich weit über die Projektgrenzen hinaus ausgedehnt. Durch die neu- und weiterentwickelten Segmentierungsmethoden wird die Auswertung der Daten deutlich erleichtert. Dabei können Klassifikationsergebnisse von bislang unerreichter Qualität erzielt werden. In Abschnitt 3.7 konnte gezeigt werden, dass das Verfahren der bisher üblichen manuellen Segmentierung sowie vergleichbaren Algorithmen in Geschwindigkeit und Genauigkeit der Ergebnisse bei einer höchstmöglichen Nutzerfreundlichkeit deutlich überlegen ist. Die intuitive Nutzung ermöglicht es, die Einstiegshürde für interessierte akademische und industrielle Gruppen drastisch zu senken. Die entwickelte Online-Anwendung und ihre Verfahren stehen als Open-Source-Software frei zur Verfügung und können ohne Weiteres in jede andere Analyse-Infrastruktur eingebettet werden. Sie sind dabei weder auf die Analyseumgebung der Antragsteller an PETRA III oder am ANKA noch auf deren Anwendung in der Morphologie beschränkt, sondern können ohne Weiteres auf andere Forschungseinrichtungen und -bereiche übertragen werden.

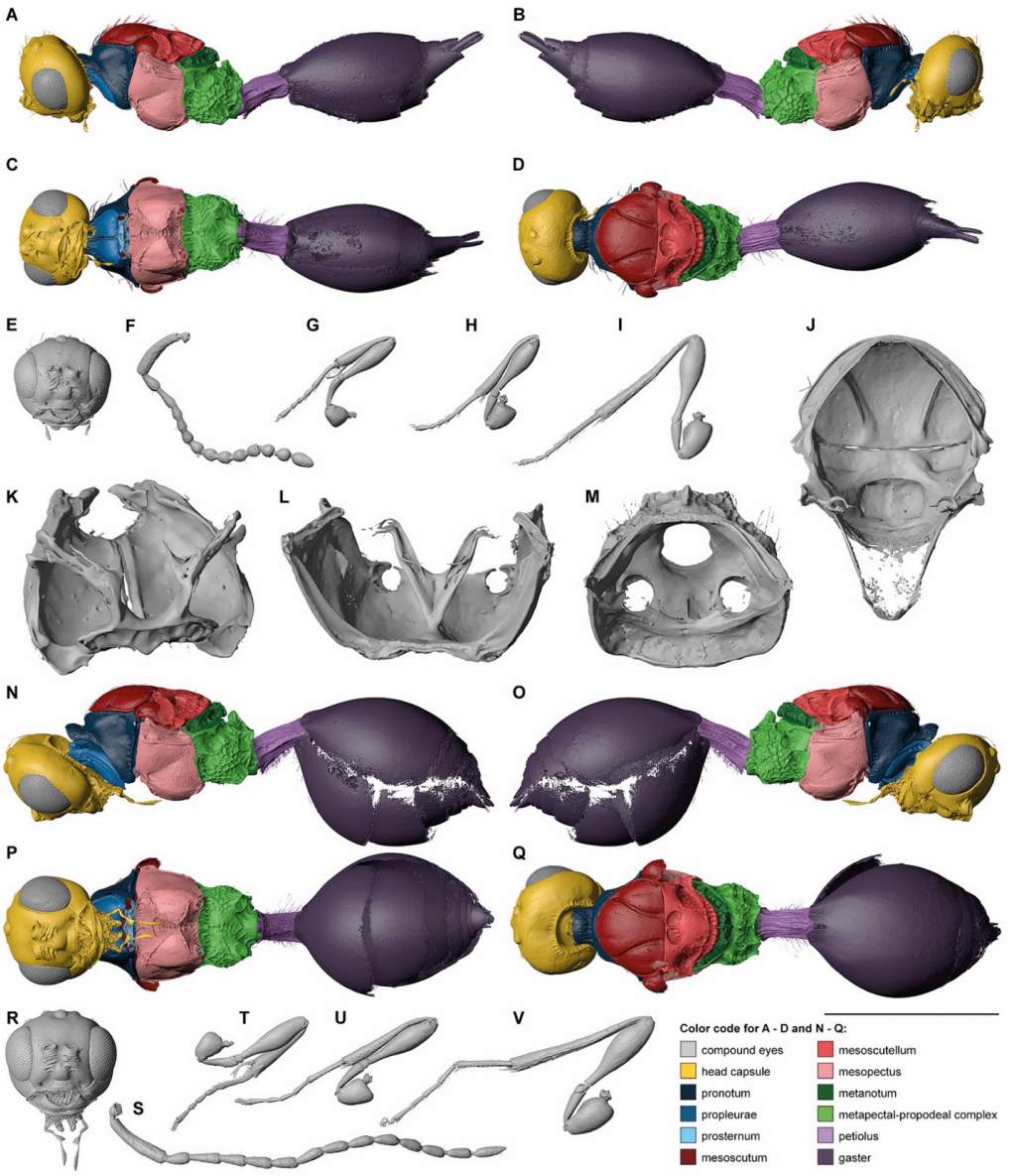

Abbildung 36: Digitale Rekonstruktion der parasitischen Wespe *Xenomorphia resurrecta* (Abschnitt 3.5.7) mit Biomedisa. Weiblicher Holotyp (**a-m**) und männlicher Paratyp (**n-v**) [van de Kamp et al., 2018].

5.1 Biomedisa Workflow

Im Folgenden wird der Arbeitsablauf der Biomedisa Online-Plattform in Kombination mit Avizo 2019.1 (Thermo Fisher Scientific, Waltham, USA) zur Segmentierung einer Vielzahl von μCT-Scans am Beispiel des Honigbienengehirns dargestellt (Abb. 38). Hierfür werden beide Hauptfeatures eingesetzt. Die GPU-basierten Random Walks (Kapitel 3) werden zur

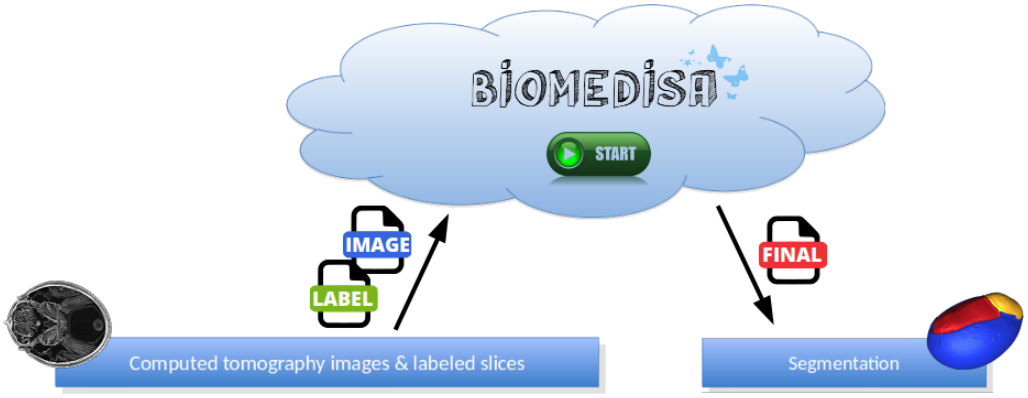

Abbildung 37: Semi-automatische Segmentierung mit der Biomedisa Online-Plattform. Die volumetrischen Bilddaten werden zusammen mit den vorsegmentierten Schichten auf die Plattform hochgeladen. Die GPU-basierte Segmentierung kann ohne Konfiguration mit einem Klick gestartet und das Ergebnis zur Weiterverarbeitung heruntergeladen werden.

Generierung von Trainingsdaten und ein künstliches neuronales Netz (Kapitel 4) zur automatisierten Segmentierung weiterer Honigbienengehirne verwendet.
Für das Training wird ein Satz vollständig segmentierter volumetrischer Bilddaten benötigt. Um die ersten drei Trainingsdatensätze zu erstellen, wurden sechs Neuropile (Mushroom Bodies (MB), zentraler Komplex (CX), Antennalloben (AL), Medullae (ME), Lobulae (LO) und andere Neuropile (OTH), Abb. 32 & 38) in jeder 10. Schicht und, falls CX enthalten ist, in jeder 5. Schicht innerhalb des 3D-Volumens von drei μCT-Scans mit Avizo 2019.1 vorsegmentiert. Anschließend wurden die GPU-basierten Random Walks verwendet, um das Volumen zwischen den vorsegmentierten Schichten zu rekonstruieren. Hierfür wurden die μCT-Daten zusammen mit den vorsegmentierten Schichten auf die Biomedisa-Plattform hochgeladen und der „Start“-Button wurde gedrückt (Abb. 37). Da die Bilddaten ein hohes Rauschen aufweisen, wurden diese vor der Interpolation durch das *Denoise*-Feature von Biomedisa (Abschnitt 5.2.6) leicht geglättet. Nach der Segmentierung wurden Ausreißer, d. h. nicht verbundene Voxel oder freistehende Inseln, automatisch entfernt (Abschnitt 5.2.2) und Segmentierungsfehler von einem/einer biologischen Experten/-in mit Avizo 2019.1 manuell korrigiert. Eine lineare Interpolation der manuell segmentierten Schichten mit Avizo 2019.1 hätte wesentlich schlechtere Ergebnisse geliefert und entsprechend mehr Korrekturaufwand benötigt. Die semi-automatische Segmentierung der 26 Trainingsbilder lieferte mit den GPU-basierten Random Walks eine um 3,9 Prozentpunkte höhere Genauigkeit gegenüber der linearen Interpolation mit Avizo 2019.1 (Tabelle 3).
Die Ergebnisse dieser drei Datensätze wurden dann genutzt, um ein künstliches neuronales Netz mit Biomedisa zu trainieren. Hierfür wurden in Biomedisa die bereits segmentierten Bilddaten zusammen mit den zugehörigen Segmentierungen ausgewählt und der „Train“-Button wurde verwendet. Nach Abschluss des Trainings erfolgte eine automatische Segmentierung weiterer Datensätze durch gleichzeitige Auswahl der Bilddaten und des trainierten Netzes sowie einem anschließenden Drückens des „Predict“-Buttons. Die auf diese Weise segmentierten Bienengehirne wurden korrigiert und den Trainingsdaten hinzugefügt.

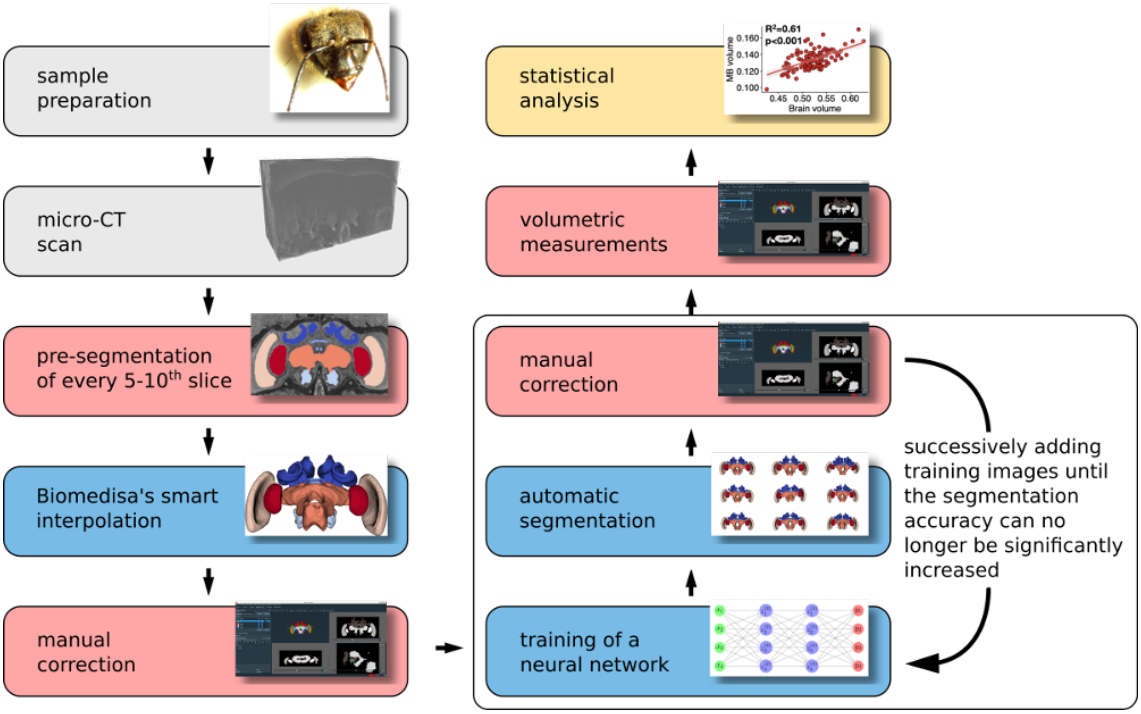

Abbildung 38: Flussdiagramm der durchgeführten Schritte zur large-scale Analyse der Gehirne von Honigbienen. Nach der Präparation der Proben und der CT-Bildaufnahmen (graue Boxen) erfolgte die Segmentierung der CT-Scans mit Avizo 2019.1 (rote Boxen) in Kombination mit Biomedisa (blaue Boxen). Abschließend wurden die Volumen mit Avizo 2019.1 bestimmt und mit R Studio (gelbe Boxen) statistisch analysiert [Lösel et al., Vorb].

Durch sukzessives Hinzufügen von weiteren Trainingsdaten, die durch das jeweils letzte künstliche neuronale Netz erstellt und anschließend manuell korrigiert wurden, wurden insgesamt fünf künstliche neuronale Netze, basierend auf 3, 7, 12, 18 und 26 Bilddaten, trainiert. Bei mehr als 26 Trainingsdaten konnte die Segmentierungsgenauigkeit nicht mehr weiter signifikant gesteigert werden (Tabelle 3). In allen Fällen wurden die Standardeinstellungen von Biomedisa verwendet (Abschnitt 4.3.1). Die Trainingszeiten unter Verwendung von 4 NVIDIA Tesla V100 betrugen 1.5, 3.5, 6, 9 bzw. 13 Stunden.
Abschließend wurde das auf 26 Bildern trainierte Netz verwendet, um weitere 84 μCT-Scans automatisch zu segmentieren. Die automatische Segmentierung eines μCT-Scans dauerte mit 1 NVIDIA GeForce GTX 1080 Ti durchschnittlich 21 Sekunden. Alle Ergebnisse wurden von einem/einer Experten/-in überprüft und bei Bedarf manuell korrigiert.
Da Biomedisa das von Avizo und Amira verwendete Amira Mesh File Format unterstützt, können die Daten problemlos zwischen der Biomedisa Online-Plattform und Avizo 2019.1 ausgetauscht werden. Für jeden Datensatz dauerte es etwa 5 bis 7 Minuten, um die Daten in Avizo 2019.1 zu importieren, auf den Bereich der Gehirne zuzuschneiden (Abb. 32d), die Daten zu exportieren, auf Biomedisa hochzuladen, die automatische Segmentierung durchzuführen, die Segmentierungsergebnisse herunterzuladen und in Avizo 2019.1 zu

importieren, um sie dort manuell zu korrigieren und die Volumen für eine anschließende statistische Analyse [Lösel et al., Vorb] zu bestimmen (Abb. 38). Je nach Qualität des automatisierten Segmentierungsergebnisses dauerte die manuelle Korrektur der Ergebnisse 1 bis 2 Minuten, bei erheblichen Fehlern jedoch deutlich länger. Typische Artefakte der automatischen Segmentierung sind Ausreißer (Abb. 33b), die entweder mit Biomedisa (Abschnitt 5.2.2) oder mit Avizo 2019.1 einfach entfernt werden können.

5.2 Biomedisa Features

Biomedisa ist eine Online-Plattform, die über einen Webbrowser zugänglich ist und keine komplexe und mühsame Konfiguration der Software oder Modellparameter benötigt. Die Anwendung ist speziell an Wissenschaftler/-innen und Mediziner/-innen ohne profunde IT-Kenntnisse gerichtet. Biomedisa unterstützt sowohl eine semi-automatische Segmentierung volumetrischer Bilddaten ohne apriorisches Wissen (Kapitel 3) als auch eine automatisierte Segmentierung einer Vielzahl ähnlicher Datensätze (Kapitel 4). Neben diesen zwei Hauptfeatures enthält die Plattform viele weitere unterschiedliche Funktionen, die über sogenannte Feature Buttons genutzt werden können, unter anderem neue Methoden zur Oberflächenglättung (Abschnitt 3.3), zum Entfernen von Ausreißern und Füllen von Löchern (Abschnitt 5.2.2) und zur Ermittlung der Unsicherheit der Segmentierungsergebnisse (Abschnitt 3.6). Die Unsicherheit kann den Benutzern/-innen Hinweise darüber geben, an welcher Stelle das Ergebnis korrigiert oder wo zusätzliche vorsegmentierte Schichten zur Initialisierung des Verfahrens eingesetzt werden sollten. Durch ihren modularen Aufbau lässt sich die Plattform leicht um weitere anwenderspezifische Funktionen erweitern (Abschnitt 5.3.3).

5.2.1 Visualisierung

Die dreidimensionale Darstellung der Segmentierungsergebnisse ist wichtig, um ein Verständnis über die inneren und äußeren Strukturen zu erhalten. Sie ist Voraussetzung für Analysen in der Funktionsmorphologie [van de Kamp et al., 2011], zur Artenbeschreibung fossiler Proben [van de Kamp et al., 2018] und unterstützt Chirurgen/-innen in ihrer Operationsplanung [Lösel and Heuveline, 2017]. Um eine Online-Visualisierung der Segmentierungsergebnisse oder ein Volumenrendering der 3D-Bilddaten zu ermöglichen, wurde die Visualisierungssoftware *ParaView Glance* in Biomedisa integriert (Abb. 40). Die Visualisierung lässt sich für einzelne Datensätze direkt durch Anklicken des jeweiligen Dateisymbols starten. Alternativ können mehrere Datensätze ausgewählt und über den „3D“ Feature Button (Abb. 39) gleichzeitig angezeigt werden. Dadurch lässt sich eine Überlagerung der Segmentierung und der zugrunde liegenden Rohdaten erzielen. Die Visualisierung von Segmentierungsergebnissen basiert auf einer durch eine Polygongrafik modellierte Annäherung der Oberfläche. Zur Berechnung des Oberflächengitters wird der

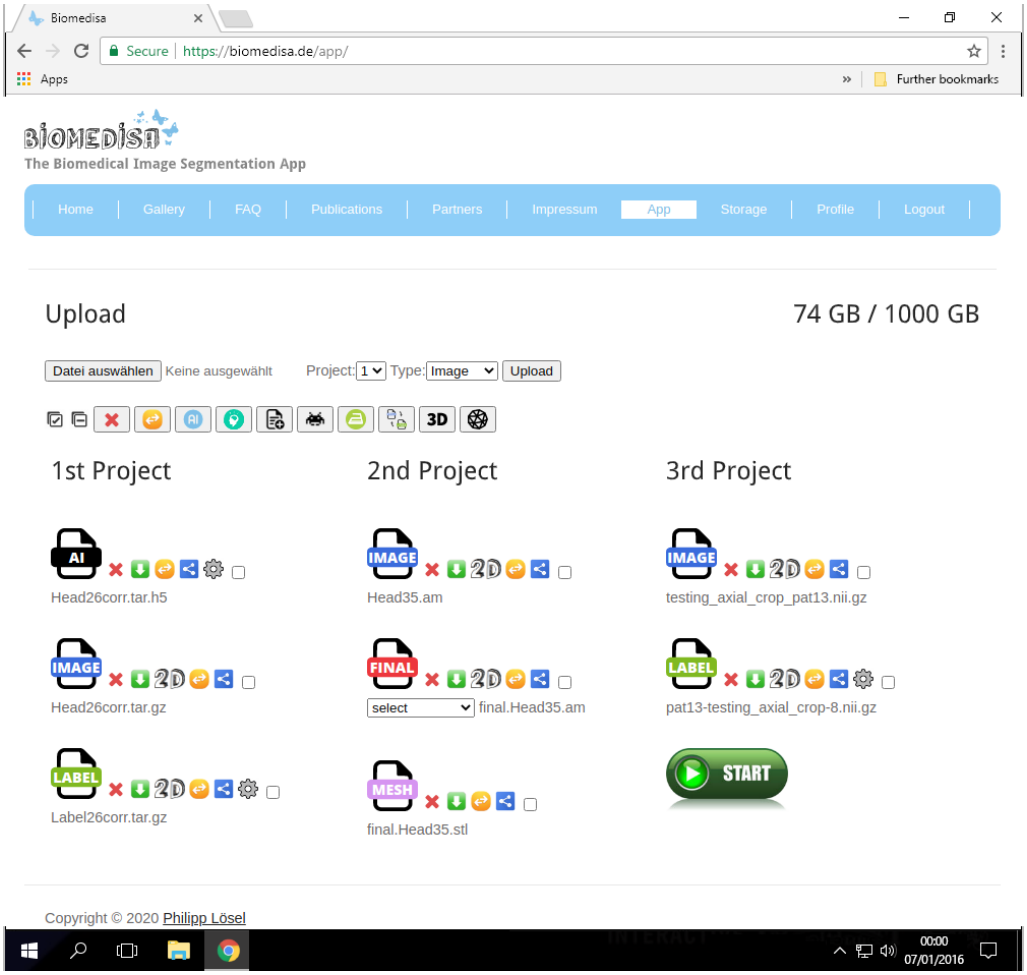

Abbildung 39: Screenshot der Biomedisa Online-Anwendung mit Feature Buttons und drei aktiven Projekten. 1st Project Abgeschlossenes Training eines künstlichen neuronalen Netzes zur automatischen Segmentierung von Honigbienengehirnen (Kapitel 4). **2nd Project** Automatische Segmentierung des Honigbienengehirns *Head35.am* durch das in **1st Project** trainierte Netz und anschließender Erzeugung eines Meshes zur dreidimensionalen Visualisierung (Abschnitt 5.2.1). **3rd Project** Semi-automatische Segmentierung eines menschlichen Herzens mittels GPU-basierter Random Walks (Kapitel 3).

Marching Cubes-Algorithmus [Lorensen and Cline, 1987] verwendet, der durch Auswahl des Segmentierungsergebnisses oder einer Label Datei und des entsprechenden Feature Buttons in Biomedisa gestartet werden kann. Dabei wird für jedes Label ein separates Oberflächengitter erzeugt und anschließend werden alle zu einem einzigen Modell zusammengefasst und in dem von *ParaView Glance* interpretierbaren STL-Format abgespeichert. Um die verschiedenen Segmente optisch zu unterscheiden, wird den generierten Dreiecken jeweils der Labelwert als Attribut zugewiesen. Anhand der Attribute lassen sich in *ParaView Glance* unterschiedliche Farbschemata auf die Oberflächen der Segmente projizieren, zum Beispiel *Linear Blue* in Abbildung 40.

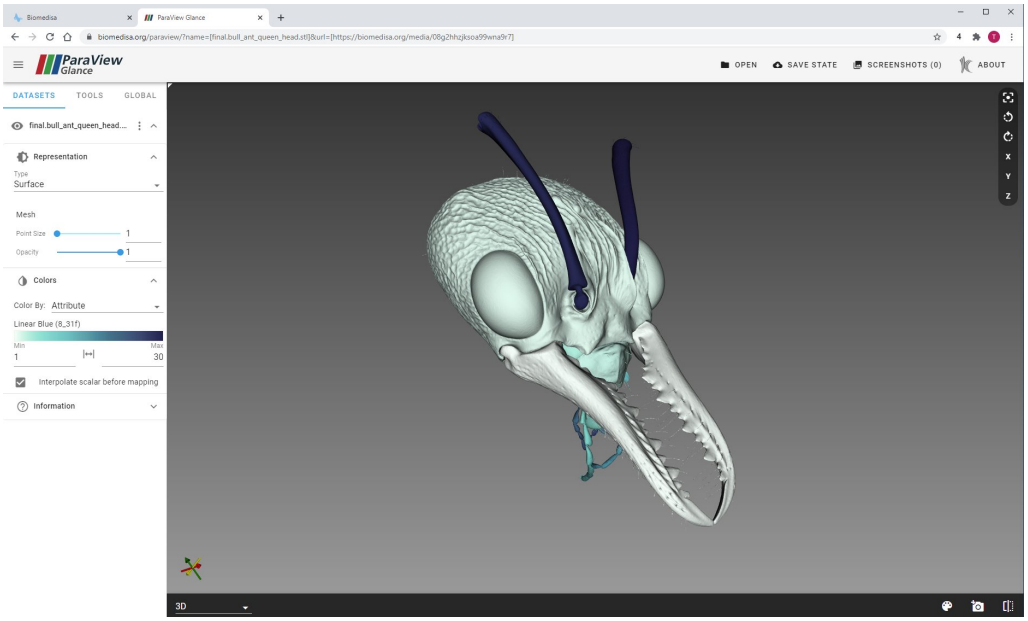

Abbildung 40: Volumengrafik des Kopfes einer australischen Bulldoggenameisenkönigin (Abschnitt 3.5.9). Die Grafik basiert auf der in Biomedisa integrierten Visualisierungssoftware *ParaView Glance* und auf einem mit Biomedisa erzeugten Volumen-Mesh.

Beim *Marching Cubes*-Algorithmus werden zunächst die Voxeldaten in Würfel (*Cubes*) unterteilt. Bei der höchsten Auflösung werden jeweils $2 \times 2 \times 2$ Voxel zu einem Würfel zusammengefasst (Abb. 41). Anschließend wird bestimmt, wie diese Würfel von der Oberfläche des Segments durchschnitten werden. Hierfür wird jeweils überprüft, ob sich die einzelnen Knoten innerhalb oder außerhalb der Segmentierung befinden. Insgesamt gibt es $2^8 = 256$ Möglichkeiten, die Ecken eines Würfels durch außen und innen liegende Voxel darzustellen. Aus Symmetriegründen reduziert sich diese Anzahl auf 15 Möglichkeiten (Abb. 41). Für jede dieser Möglichkeiten werden Dreiecke definiert, die den Schnitt der Oberfläche modellieren und die Würfel in außen und innen liegende Bereiche unterteilen. Die Dreiecke werden für jede mögliche Kombination in einer *Triangle Lookup* Tabelle gespeichert. Nachdem für jeden Würfel die entsprechenden Dreiecke bestimmt wurden, kann daraus das Oberflächengitter konstruiert werden. Zudem verfügt Biomedisa über einen integrierten *Slice Viewer*. Dieser ermöglicht dem/der Benutzer/-in einen Einblick in jede Schicht der Ergebnisse oder der Bilddaten zu erhalten. Zur Darstellung der Segmentierungsergebnisse in den Schichten wird jedem Segment eine Farbe zugewiesen und die Kanten der Segmente als Kontur in den Bilddaten hervorgehoben (Abb. 23 links & 33 links). Der *Slice Viewer* kann über die „2D“ Feature Buttons neben den Dateien gestartet werden. Die Projektionen der Kanten werden unmittelbar nach der Segmentierung erstellt. Zur Reduzierung des Datentransfers werden nur die Schichten an den/die Nutzer/-in übertragen, die beim Durchlaufen der Bildschichten angezeigt werden.

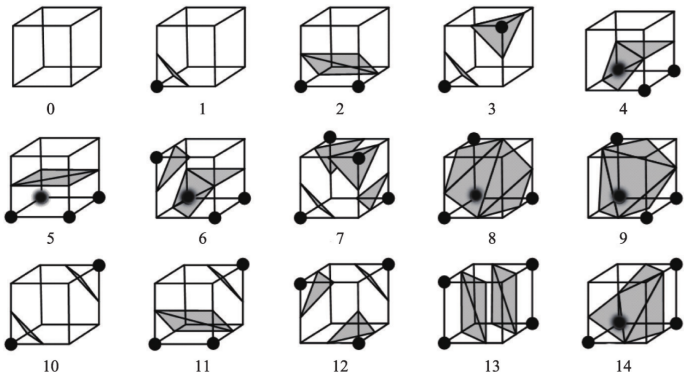

Abbildung 41: Mögliche Schnitte der Oberfläche durch einen Würfel im *Marching Cubes*-Algorithmus zur Erzeugung eines Polygongitters. Die 256 möglichen Fälle reduzieren sich aufgrund von Symmetriegründen auf 15 Fälle. Die markierten Ecken befinden sich innerhalb des Segments, die anderen außerhalb [Wang et al., 2021].

5.2.2 Entfernen von Ausreißern und Füllen von Löchern

Um mögliche Ausreißer (nicht verbundene Voxel oder Inseln) zu entfernen, wurde in Biomedisa ein Verfahren integriert, das zuerst alle freistehenden Objekte der Segmentierung identifiziert und dann die Objekte entfernt, deren Volumen kleiner als ein vordefinierter Schwellenwert ist. In der Standardkonfiguration beträgt dieser Schwellenwert 10 % des größten identifizierten Objekts. Durch die Einführung eines Schwellenwerts soll verhindert werden, dass einzelne Teile von Objekten, die sich aus mehreren Einzelteilen zusammensetzen, entfernt werden.
Bei dieser Methode wird allen Voxel, die zu einem Segment gehören, zunächst ihr Index als Wert zugewiesen. Dann wird von links nach rechts, von oben nach unten und von der ersten bis zur letzten Schicht über die Voxel iteriert, wobei jedem Voxel der kleinste positive Wert seiner direkten Nachbarn zugewiesen wird. Dieser Vorgang wird solange wiederholt, bis keine neue Zuordnung erfolgt. Dadurch entstehen voneinander getrennte Einzelobjekte mit jeweils einem einheitlichen Wert, die anschließend entsprechend ihrer Größe beibehalten oder gelöscht werden. Die gleiche Technik wird verwendet, um Löcher in den verschiedenen Segmenten zu füllen. In der Standardkonfiguration werden Löcher gefüllt, sobald sie kleiner als 90 % der Gesamtgröße des zugehörigen Segments sind.

5.2.3 Dateiformate & Datentypen

Die folgenden dreidimensionalen Dateiformate werden unterstützt: Multipage TIFF, Amira Mesh (AM), MHD, MHA, NRRD und NIfTI (.nii & .nii.gz). Darüber hinaus kann auch ein gezippter Ordner (ZIP) hochgeladen werden, der zweidimensionale Schichten als DICOM, PNG oder TIFF enthält, die das Volumen repräsentieren. Für das Training eines neurona-

len Netzes können die Trainingsdaten zusammen mit den korrespondierenden Label einzeln ausgewählt oder alle Bilddaten und alle Label-Dateien jeweils in einem TAR-Archiv auf Biomedisa hochgeladen werden. Das Hochladen der Dateien in Form eines TAR-Archivs ermöglicht beliebig viele Trainingsdaten, wohingegen die Auswahl einzelner Dateien auf die maximal mögliche Anzahl von neun Projekten beschränkt ist. Um eine eindeutige Zuordnung zu garantieren, müssen die Dateinamen der jeweiligen TAR-Archive übereinstimmen. Eine mögliche Kombination wäre *heart1.tif - heart9.tif* für die Bilddaten und *label.heart1.tif - label.heart9.tif* für die Label-Daten.
Die Metainformationen der ursprünglichen Label-Dateien werden übernommen. Zum Beispiel bleiben Labelnamen und Labelfarben erhalten, so dass das Ergebnis ohne Weiteres in das präferierte Segmentierungstool des/der Benutzers/-in importiert und dort weiterverarbeitet werden kann.
Die Bilddaten können als Ganz- oder Fließkommazahlen mit 8-Bit, 16-Bit, 32-Bit oder 64-Bit verarbeitet werden. Es gibt zwei Datenverarbeitungsverfahren: Um den Speicherverbrauch zu reduzieren, werden 8-Bit-Bilder separat verarbeitet (Abschnitt 3.2.1), während 16-Bit-, 32-Bit- und 64-Bit-Bilder in ein Gleitkommaformat mit einfacher Genauigkeit konvertiert und auf das Intervall von 0 bis 255 skaliert werden. Vor Beginn der Segmentierung können Bilder mithilfe eines Feature Buttons in ein 8-Bit Multipage TIFF konvertiert werden. Dies ist nützlich, wenn der verfügbare GPU-Speicher nicht ausreicht, um sehr große Datensätze, die nicht in 8-Bit vorliegen, zu verarbeiten. Darüber hinaus können die Daten zwischen Nutzern/-innen über die jeweiligen Benutzernamen geteilt werden. Zusätzlich kann ein passwortgeschützter Downloadlink für die einzelnen Datensätze erstellt werden.

5.2.4 Nachbearbeitung mit aktiven Konturen

In Biomedisa wird das Segmentierungsergebnis der GPU-basierten Random Walks automatisch mit aktiven Konturen (Abschnitt 2.1) nachbearbeitet und das Ergebnis als optionales Segmentierungsergebnis zur Verfügung gestellt. Die numerische Berechnung der aktiven Konturen, und insbesondere deren Berechnung in drei Raumdimensionen, ist in der Regel sehr aufwendig und dauert dementsprechend lange. Zudem wird bei dieser Methode immer nur eine lokale, aber nicht notwendigerweise globale Lösung berechnet. Dies ist insbesondere dann problematisch, wenn die initial gesetzte Kontur sich sehr weit von der gesuchten Lösung entfernt befindet und das Verfahren dadurch leicht an einem lokalen Minimum terminiert bevor die Kontur die gewünschte Lösung erreicht hat.
Jedoch hat sich der in Abschnitt 2.1 vorgestellte Ansatz einer aktiven Kontur, die nicht auf dem Bildgradienten basiert, insbesondere bei homogenen Objekten bewährt. Die Kombination der semi-automatischen Segmentierung durch GPU-basierte Random Walks mit einer Nachbearbeitung mittels aktiver Konturen kann in manchen Fällen ein besseres Endergebnis liefern. Dabei dient die Segmentierung durch Random Walks zunächst als erste Näherung an die gesuchte Lösung, welche dann durch eine Nachbearbeitung mittels aktiver Konturen in wenigen Iterationsschritten verfeinert werden kann.
Der hohe Rechenaufwand beim Lösen der partiellen Differentialgleichung 2.3 entsteht durch

den enthaltenen Krümmungsterm. In diesem Abschnitt soll darum ein morphologischer Operator vorgestellt werden, dessen sukzessive Anwendung den in der Gleichung 2.3 enthaltenen mittleren Krümmungsfluss 3.10 approximiert und dadurch eine schnellere Berechnung der Konturentwicklung ermöglicht [Marquez-Neila et al., 2014].

Definition 5.1 (Dilatation & Erosion)**.** *Die Dilatation D_h mit radius h einer Funktion $u\colon \mathbb{R}^d \to \mathbb{R}$ sei definiert als*

$$D_h u(x) \coloneqq \sup_{y \in hB(0,1)} u(x+y),$$

und die Erosion sei definiert als

$$E_h u(x) \coloneqq \inf_{y \in hB(0,1)} u(x+y),$$

wobei $B(0,1)$ eine d-dimensionale Sphere mit Radius 1, zentriert um $\vec{0}$ und hB die um h skalierte Menge $hB = \{hx \mid x \in B\}$.

Proposition 5.2. *Jeder morphologische Operator T kann mithilfe von Supremum und Infimum als sup-inf Operator*

$$(T_h u)(x) = \sup_{B \in \mathcal{B}} \inf_{y \in x+hB} u(y),$$

oder als inf-sup Operator

$$(T_h^* u)(x) = \inf_{B \in \mathcal{B}} \sup_{y \in x+hB} u(y)$$

dargestellt werden, wobei $\mathcal{B}$ eine Basis des Operators ist. Für $\mathcal{B} = \{B(0,1)\}$ ergibt sich entsprechend $(T_h^* u)(x) = D_h u(x)$ und $(T_h u)(x) = E_h u(x)$.

Satz 5.3. *Sei eine d-dimensionale Hyperfläche implizit gegeben durch die Funktion $u\colon \mathbb{R}^d \to \mathbb{R}$ mit $(d-1)$-dimensionalen Niveauflächen. Des Weiteren seien die morphologischen Operatoren SI_h und IS_h die sup-inf und inf-sup Operatoren zur Basis $\mathcal{B}^d = \{K_n \mid n \in S^{d-1}\}$, wobei $S^{d-1} = \{n \in \mathbb{R}^d \mid \|n\| = 1\}$ und $K_n = \{v \in \mathbb{R}^d \mid \|v\| \leq 1, v^T n = 0\}$. Dann gilt für den d-dimensionalen Operator*

$$(F_h u)(x) \coloneqq \frac{(SI_{2h} u)(x) + (IS_{2h} u)(x)}{2}$$

das Grenzwertverhalten

$$\lim_{h \to 0^+} \frac{(F_{\sqrt{h}} u) - u}{h} = \left(\min(\kappa_1, \ldots, \kappa_{d-1}, 0) + \max(\kappa_1, \ldots, \kappa_{d-1}, 0)\right) |\nabla u|,$$

wobei κ_i für $i = 1, \ldots, d-1$ die Hauptkrümmungen der Fläche u sind.

Beweis. [Marquez-Neila et al., 2014, Theorem 3.3]. □

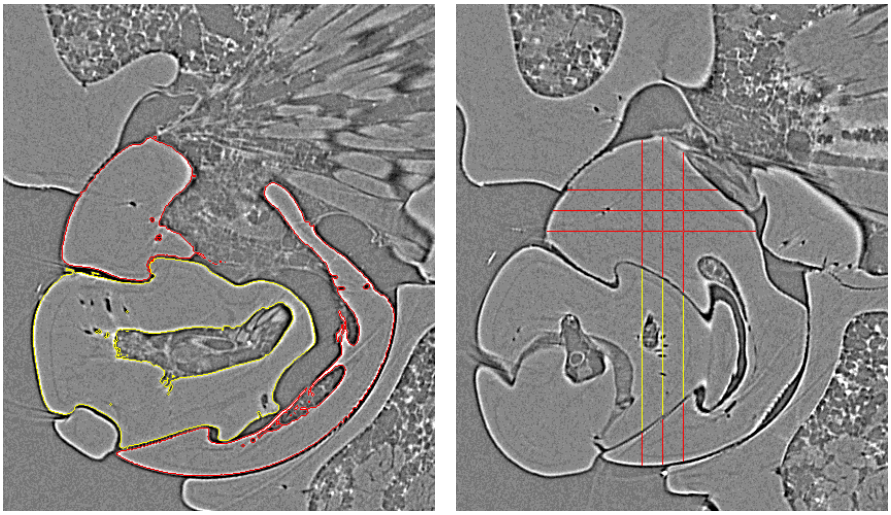

Abbildung 42: Vorsegmentierung mit verschiedenen Ausrichtungen zur Segmentierung eines biologischen Schraubengelenks. Das *All axes* Feature ermöglicht neben der Vorsegmentierung in der xy-Ebene (**links**) auch andere Ausrichtungen. In der Standardausrichtung werden diese als horizontal und vertikal verlaufende Linien sichtbar (**rechts**).

Folgerung 5.4. *Sei* $d = 3$ *und für die Hauptkrümmungen* κ_1, κ_2 *gilt* $\operatorname{sgn}(\kappa_1) \neq \operatorname{sgn}(\kappa_2)$*, dann gilt für den Operator* F_h*:*

$$\lim_{h \to 0^+} \frac{(F_{\sqrt{h}} u) - u}{h} = - |\nabla u| \operatorname{div}\left(\frac{\nabla u}{|\nabla u|}\right).$$

Beweis. Die Aussage folgt unmittelbar aus Satz 5.3, Definition 3.20 und Satz 3.22. □

Nach [Marquez-Neila et al., 2014, Lemma 3.1] approximiert der morphologische Operator $SI_{\sqrt{h}} \circ IS_{\sqrt{h}}$ für kleines h den Operator $F_{\sqrt{h}}$. Die sukzessive Anwendung des morphologischen Operators $SI_{\sqrt{h}} \circ IS_{\sqrt{h}}$ kann folglich verwendet werden, um die Gleichung 3.10 sowie den Krümmungsterm in der partiellen Differentialgleichung 2.3 zu approximieren. Dabei muss jedoch beachtet werden, dass für $\operatorname{sgn}(\kappa_1) = \operatorname{sgn}(\kappa_2)$ die Abweichung umso größer ausfällt, je stärker $\min(|\kappa_1|, |\kappa_2|)$ von Null abweicht.

5.2.5 Vorsegmentierung mit verschiedenen Ausrichtungen

Die Vorsegmentierung erfolgt in den meisten Fällen in der Standardausrichtung. Dabei verläuft die xy-Ebene horizontal und die z-Achse vertikal. Wird nicht ausschließlich die Standardausrichtung zur Vorsegmentierung einzelner Schichten verwendet, muss in den

Biomedisa-Einstellungen die Funktion *All axes* aktiviert werden. Wird diese Funktion aktiviert, muss in jeder Ausrichtung mindestens eine Schicht zwischen zwei aufeinanderfolgenden vorsegmentierten Schichten freigelassen werden, so dass die Schichten als horizontal oder vertikal verlaufende Linien in den jeweils anderen Ausrichtungen erkennbar sind (Abb. 42). Zur Identifikation der vorsegmentierten Schichten müssen für jede der möglichen Ausrichtungen zunächst alle horizontal und vertikal verlaufenden Linien entfernt werden, so dass lediglich die in der jeweiligen Ausrichtung vorsegmentierten Schichten übrig bleiben.

5.2.6 Glätten von Bilddaten

Die volumetrischen Bilddaten $u\colon \mathbb{R}^3 \to \mathbb{R}$ können durch den Medianfilter

$$F(x,y,z) = \frac{1}{|M|}\left(\sum_{(i,j,k)\in M} u(i,j,k)\right),$$

mit einer Filtermaske M der Größe $3 \times 3 \times 3$ Voxel geglättet werden. Da die μCT-Scans der Honigbienengehirne (Abb. 33) ein hohes Rauschen aufweisen, konnte bei der Erstellung der Trainingsdaten (Abschnitt 5.1) durch das vorherige Glätten der Bilddaten eine Genauigkeitssteigerung von 1,01 % für die GPU-basierten Random Walks erreicht werden.

5.3 Plattform & Infrastruktur

Biomedisa ist ein dynamischer Webservice, der auf einer LAMJ-Umgebung aufbaut. Als Betriebssystem dient eine Linux Distribution (L), der Webserver, der die Seite über das World Wide Web bereitstellt, ist ein Apache-Server (A) und als Datenbank zum Abspeichern der anwendungsbezogenen Daten wird ein MySQL-Datenbank-Managementsystem (M) verwendet. Die genutzten Programmiersprachen sind JavaScript (J) und Python. Dabei stellt das Webframework Django die Grundlage für Biomedisa dar. Django ermöglicht eine explizite Konfiguration der Website über URLs (*Uniform Resource Locator*). Das heißt, jede URL kann einer Python-Funktion zugeordnet werden und diese steuern. Des Weiteren verwendet Django eine objektrelationale Abbildung, um direkt die Einträge der MySQL Datenbank zu verarbeiten. Die visuelle Darstellung der Online-Plattform erfolgt über HTML-Dateien.

5.3.1 Biomedisa Rechencluster

Bei der Server Architektur des Biomedisa Clusters wird eine Strategie der Komplettreplikation auf mehreren Servern und mehreren Rechenzentren verfolgt (Abb. 43). Dieser

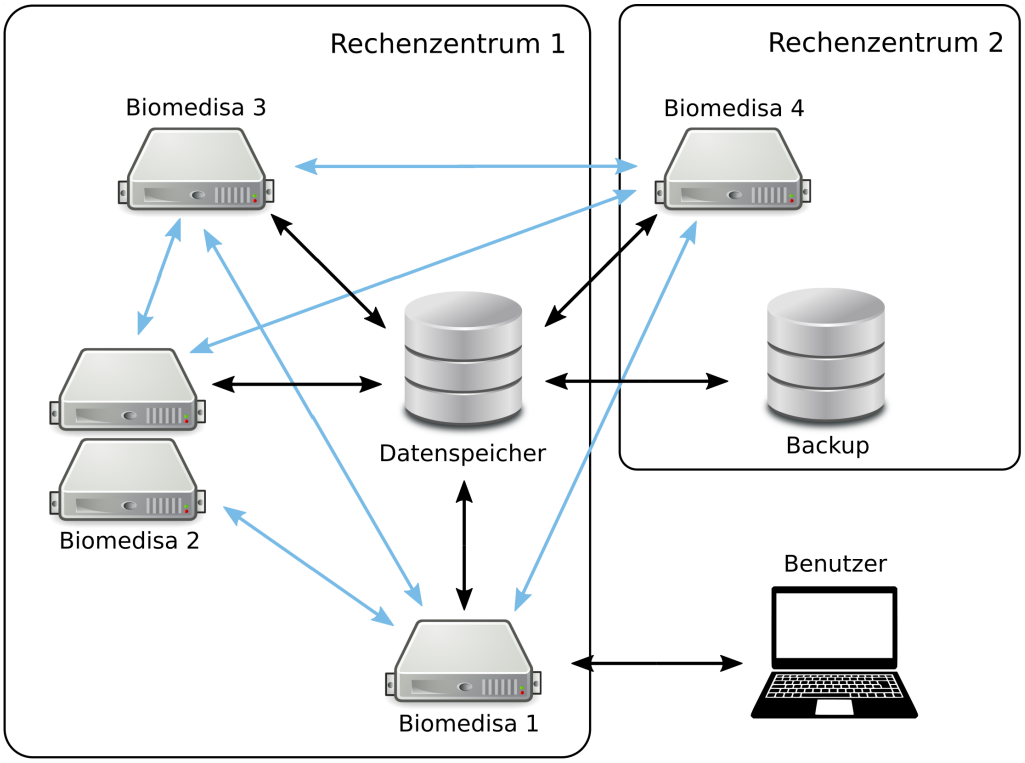

Abbildung 43: Biomedisa Rechencluster. Die Rechen- und Speicherinfrastruktur der Biomedisa Online-Plattform verteilt sich auf mehrere Rechenknoten (Biomedisa 1-4) und mehrere Rechenzentren (Rechenzentrum 1 & 2). Einzelne Rechenknoten können aus einem Zusammenschluss mehrerer Server bestehen (Biomedisa 2). Die Datenbanken aller Rechenknoten sind über eine MySQL Master-Master-Replikation miteinander verbunden (blaue Pfeile). Es gibt einen zentralen Datenspeicher, der auf jedem Rechenknoten gemountet ist (schwarze Pfeile) und ein zugehöriges Backup System. Ein beliebiger Knoten dient als Schnittstelle für die Benutzer/-innen.

dezentrale Aufbau ermöglicht eine hohe Variabilität und eine sogenannte *Desaster Recovery*. Das System bleibt selbst bei einem Totalverlust eines Rechenzentrums komplett funktionsfähig. Die Online-Plattform wird dabei in identischer Weise auf einer beliebigen Anzahl von Servern installiert. Einer dieser Server dient als Login-Knoten und somit als Schnittstelle für die Benutzer/-innen der Plattform. Aufgrund der Konfiguration des Gesamtsystems kann dieser Server beliebig gewählt und jederzeit gewechselt werden. Der gewählte Login-Knoten verteilt alle eingehenden Aufgaben gleichmäßig auf die Rechenknoten. Durch diesen Ansatz ist es jederzeit möglich, weitere Rechenknoten in das System zu integrieren oder vorhandene zu entfernen. Einzelne Rechenknoten können aus einem Verbund von mehreren Servern bestehen, um Berechnungen mit einer großen Anzahl von GPUs zu ermöglichen (Abschnitt 3.2). Die MySQL-Datenbanken aller Rechenknoten sind über eine Master-Master-Replikation miteinander verbunden (Abschnitt 5.3.2).
Die Nutzerdaten befinden sich auf einem zentralen Speicher, der von allen Servern gemountet wird. Der Datenspeicher basiert auf einem RAID-System (*Redundant Array of*

Independent Disks) des Levels 5 [Patterson et al., 1988]. Dabei werden mehrere physische Festplatten zu einem virtuellen, logischen Laufwerk zusammengeschlossen. Dieses Laufwerk besitzt einen Speicherplatz der Größe $k(n-1)$, wobei k der Speicherplatz jeder einzelnen Festplatte ist und n die Anzahl der verwendeten Festplatten. Bei diesem RAID-System kann eine Festplatte ausfallen, ohne dass es zu einem Datenverlust kommt. Die Daten werden dabei in sogenannten *Stripes* auf die Festplatten geschrieben. Jeder *Stripe* besteht aus Blöcken, die sich auf die Festplatten verteilen (Tabelle 4). Durch das Verteilen der Blöcke auf mehrere Festplatten können die Daten parallel eingelesen werden, wodurch sich eine erhöhte Lesegeschwindigkeit ergibt. Zur Erzeugung einer Festplattenredundanz werden für jeden *Stripe* aus den Daten sogenannte Paritätsinformationen berechnet und ebenfalls abgespeichert (Tabelle 4). Die Paritätsinformationen sind das Ergebnis der XOR-Verknüpfung

$$x \text{ XOR } y :\Leftrightarrow ((\neg x \wedge y) \vee (x \wedge \neg y)), \tag{5.1}$$

mit

x	y	x XOR y
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

Die XOR-Funktion 5.1 verknüpft zwei Werte derart, dass sich eine logische 1 ergibt, wenn die Eingangsgrößen verschieden sind. Zur Berechnung der Paritätsinformationen werden die Blöcke eines *Stripes* nacheinander XOR-verknüpft. Beispielsweise ergibt sich in Tabelle 4 für die Festplatten 1 & 2 aus den Blöcken des ersten *Stripes* (0100 & 0101) der Block 0001. Zusammen mit dem Block der 3. Festplatte 0010 ergibt sich die Paritätsinformation 0011, die auf der 4. Festplatte abgespeichert wird. Fällt nun eine Festplatte aus, dann lassen sich die verlorengegangenen Blöcke aus diesen Informationen rekonstruieren. Ein RAID-5 benötigt hierfür mindestens drei Festplatten. Nehmen wir beispielsweise an, dass Festplatte 1 verloren geht. Für *Stripe* 1 ergibt 0101 XOR 0010 (Platte 2 und Platte 3) den Block 0111. Der Wert 0111 XOR 0011 (Platte 4) ergibt 0100. Entsprechend ließen sich auch die übrigen Blöcke der *Stripes* 2-4 berechnen. Fällt jedoch mehr als eine Platte aus, dann gehen alle Daten verloren. Ein weiterer Nachteil dieses Verfahrens ist eine geringere Schreibgeschwindigkeit, da zusätzlich die Paritätsdaten berechnet und abgespeichert werden müssen. Durch das Abspeichern der Paritätsdaten ergibt sich zudem ein leichter Kapazitätsverlust. Außerdem ist das Wiederherstellen der Daten nach einem Festplattenausfall äußerst aufwendig. Da das RAID-System nur eine Ausfallsicherheit gegenüber dem Verlust einer Festplatte bietet, werden die Daten in Biomedisa zusätzlich mit einem weiteren Speichersystem synchronisiert, das über verschiedene Backup-Level verfügt und sich in einem anderen Rechenzentrum befindet (Abb. 43).

Tabelle 4: Paritätsinformationen in einem RAID-5-Verbund. Die Paritätsinformationen sind violett hervorgehoben. Sie ergeben sich durch XOR-Verknüpfungen der anderen Blöcke des selben *Stripes*.

	Platte 1	**Platte 2**	**Platte 3**	**Platte 4**
Stripe 1	0100	0101	0010	0011
Stripe 2	0010	0000	0110	0100
Stripe 3	0011	0001	1010	1000
Stripe 4	0110	0001	1101	1010

5.3.2 MySQL Master-Master-Replikation

Eine klassische Replikation einer MySQL-Datenbank besteht aus einem primären Server (*Primary* oder *Host*) und einem oder mehreren sekundären Servern (*Replicas* oder *Worker*). Die MySQL-Replikation ermöglicht es, mehrere Kopien einer Datenbank auf unterschiedlichen Servern abspeichern zu können, wobei die Synchronisation der Daten automatisch erfolgt. Daten auf dem Host, wie Dateinamen, Bearbeitungsstatus oder Konfigurationsparameter der Prozesse, werden auf einen oder mehreren Workern repliziert. Alle Änderungen auf dem Master können dabei automatisch von den Workern übernommen und in deren MySQL-Datenbanken geschrieben werden.
Die Replikation ermöglicht es zum einen Ausfallzeiten zu reduzieren, zum Beispiel bei der Wartung des Systems oder eines Serverausfalls, und zum anderen die Ressourcen des Hosts zu entlasten, zum Beispiel um regelmäßige Backups oder Datenanalysen durchführen zu können (Entlastung). Darüber hinaus ermöglicht die Verteilung der Daten auf verschiedene Server und verschiedene Rechenzentren eine Absicherung gegen einen Totalausfall des Systems und einen Totalverlust der Daten (*Desaster Recovery*). Des Weiteren werden bei Zugriffen auf die Datenbanken deren Einträge sehr häufig nur ausgelesen und es erfolgt kein Schreiben in die Datenbank. Solche Abfragen können ohne Weiteres auf mehrere Server verteilt werden, was bei einem sehr hohem Besucheraufkommen sogar unumgänglich ist (Lastverteilung). Bei der asynchronen Replikation schreibt der Host alle Änderungen an der Datenbank (Anlegen neuer Tabellen, Speichern von neuen oder Ändern von bestehenden Werten, Löschen von Zeilen etc.) in eine binäre Logdatei (Binlog, *Binary Log Files*), wodurch alle Veränderungen protokolliert werden. Diese Transaktionsprotokolle werden an die Worker geschickt, sobald zu diesen eine Verbindung besteht, und dort in deren Datenbank übernommen. Der Host arbeitet unmittelbar weiter, ohne auf eine Bestätigung der Worker zu warten. Um die Konsistenz und Datenintegrität zu erhöhen, ist es möglich, dass der Host bei einer semi-synchronen Replikation solange mit neuen Schreibvorgängen wartet, bis alle Worker den Erhalt und die Änderungen an ihrer Datenbank bestätigt haben, was jedoch mit gewissen Performance-Einbußen einhergeht. Die Binlogs enthalten sogenannte *Global Transaction Identifiers* (GTIDs) mit universell eindeutigen Identifikationsnummern (UUIDs), die jede Transaktion auf dem Server und innerhalb des Replikations-Netzwerks eindeutig identifizieren. Dadurch lässt sich die Kommunikation zwischen den Servern nachverfolgen und bei einem Ausfall das System wiederherstellen. Biomedisa nutzt für seine

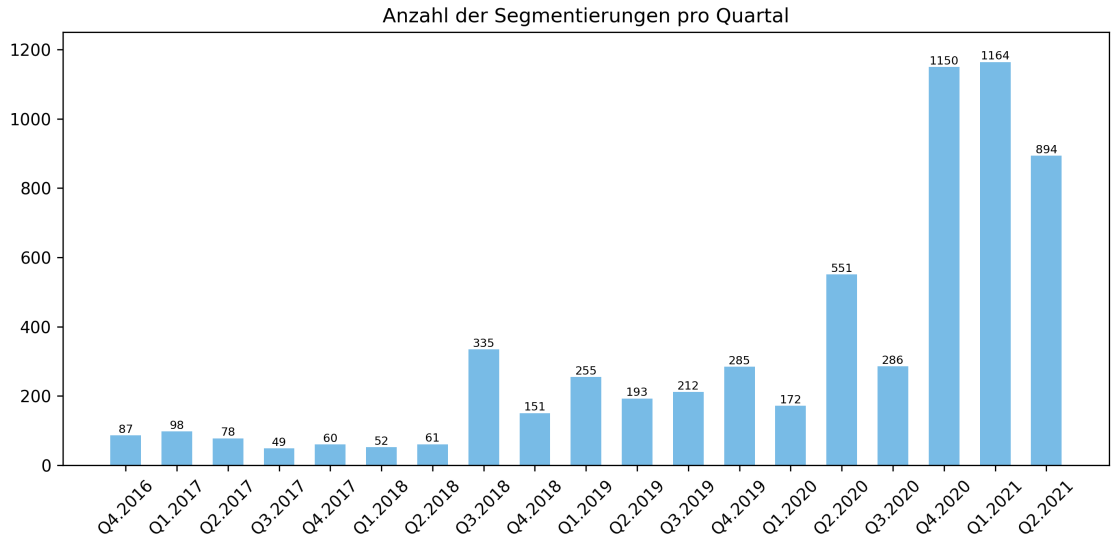

Abbildung 44: Durchgeführte Segmentierungen auf der Biomedisa Online-Plattform. Anzahl der Segmentierungen pro Quartal vom Start der Plattform im Oktober 2016 bis Juni 2021.

Strategie der Komplettreplikation eine sogenannte Master-Master-Replikation zwischen je zwei Servern. Das heißt, ein Server erfüllt für alle anderen Server im Cluster immer sowohl die Funktion des Hosts als auch die Funktion des Workers (Abb. 43). Für die Master-Master-Replikation können für eine geringere Latenz und höhere Bandbreite auch direkte Verbindungen zwischen den Servern mit InfiniBand oder 10-Gigabit Ethernet verwendet werden.

5.3.3 Modularisierung & Workload Management

Auf jedem Server befindet sich zu jedem Feature bzw. Modul (Abschnitt 5.2) ein Worker und eine *Queue* (Warteschlange). Eingehende Aufgaben werden zunächst in den entsprechenden *Queues* aufgereiht und dann sukzessive von dem zugehörigen Worker abgearbeitet. Bearbeitet ein Server bereits einen Prozess, dann werden zusätzliche Aufgaben für eine ausgewogene Lastverteilung (*Workload Balance*) gleichmäßig auf die entsprechenden *Queues* der anderen Rechenknoten im Biomedisa Cluster (Abschnitt 5.3.1) verteilt. Die Worker arbeiten dabei asynchron. Dadurch müssen die Webserver nicht auf den Abschluss der jeweiligen Prozesse warten und können unmittelbar weiterarbeiten. Die Modularisierung erleichtert nicht nur die Verteilung der Prozesse auf verschiedene Rechenknoten, sondern auch das Hinzufügen von neuen Features. Hierfür können diese durch eine Python-Funktion definiert und über einen zusätzlichen Feature-Button aufgerufen werden. Grundsätzlich können jedem Modul mehrere Worker oder jedem Worker mehrere *Queues* zugeordnet werden. Es

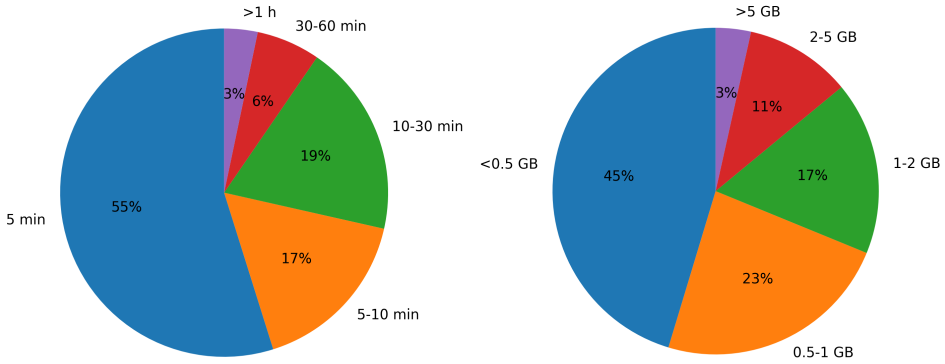

Abbildung 45: Aufteilung der durchgeführten Segmentierungen auf der Biomedisa Online-Plattform. links Nach Segmentierungsdauer. **rechts** Nach Größe der Datensätze.

ist jedoch ratsam, die absolute Anzahl verarbeitender Prozesse möglichst klein zu halten, da die volumetrischen Bilddaten und die bei der Verarbeitung anfallenden Prozessdaten sehr schnell viel Arbeitsspeicher in Anspruch nehmen. Die Modularisierung der Features in Kombination mit der Master-Master-Replikation (Abschnitt 5.3.2) ermöglicht eine Arbeitslastverteilung innerhalb des Rechenclusters, die problemlos um zusätzliche Module und Rechenkapazitäten erweitert werden kann.

5.4 Nutzerstatistiken

Im August 2021 hatte Biomedisa 486 registrierte Nutzer/-innen, von denen 236 mindestens eine Segmentierung durchgeführt haben. Insgesamt wurden bis dahin 6460 Segmentierungen getätigt, wobei die Zahl der durchschnittlichen Segmentierungen pro Quartal eine stark steigende Tendenz aufweist (Abb. 44). Der/die Nutzer/-in mit den meisten Segmentierungen führte insgesamt 712 erfolgreiche Segmentierungen durch. Eine Segmentierung dauerte im Durchschnitt 15 min 22 s. Etwas mehr als die Hälfte aller Segmentierungen dauerte weniger als 5 min. Lediglich 3 % der Segmentierungen dauerten länger als 1 h (Abb. 45 links). Die Größe der segmentierten Bilder wurde erst ab dem zweiten Quartal 2018 aufgezeichnet (Abb. 45 rechts). Ab diesem Zeitpunkt hatten 45 % der segmentierten Daten eine Größe von weniger als 500 MB. Mehr als ein Viertel der Daten wies eine Größe von mehr als 1 GB auf. Es wurden 136 Datensätze mit einer Größe von 5 GB oder mehr segmentiert. Das größte verarbeitete Bild hatte eine Größe von etwa 40 GB. Dabei ist jedoch zu beachten, dass Biomedisa das Bild vor der Verarbeitung automatisch auf den gewünschten Bereich zuschneidet (Abschnitt 3.5.11). Das größte verarbeitete Bild, das nicht zugeschnitten wurde, war 16 GB groß.

6 Diskussion & Ausblick

In dieser Dissertation wurde ein nutzerfreundliches und universell anwendbares Verfahren entwickelt, um die mühsame, fast ausschließlich manuelle Segmentierung, die nach wie vor in vielen wissenschaftlichen Disziplinen die 3D-Bildanalyse dominiert, zu ersetzen. Es konnte gezeigt werden, dass das neu entwickelte und in [Lösel and Heuveline, 2016, Lösel et al., 2020] vorgestellte GPU-basierte Verfahren (Abschnitt 3) die übliche Vorgehensweise zur Segmentierung sehr großer volumetrischer Bilddaten unbekannter Morphologie deutlich erleichtert, beschleunigt sowie die Qualität der Segmentierungsergebnisse signifikant verbessert (Abschnitt 3.4 & Abb. 17). Im Gegensatz zu der weit verbreiteten traditionellen Methode einer sehr dichten manuellen Vorsegmentierung vieler einzelner Schichten mit anschließender linearer Interpolation und manueller Korrektur, werden bei dem GPU-basierten Ansatz zusätzlich die Bilddaten für die Segmentierung des Volumens berücksichtigt. Die Berücksichtigung der Bildinformationen lässt wesentlich größere Abstände zwischen den vorsegmentierten Schichten zu und reduziert somit den insgesamt benötigten manuellen Arbeitsaufwand.

Biomedisa ist deutlich effizienter als die traditionelle Vorgehensweise

Für eine akkurate Segmentierung eines *Trigonopterus* Rüsselkäfers wurden für dieses Verfahren lediglich 37 manuell vorsegmentierte Schichten benötigt (Abschnitt 3.4). Die Wahl dieser Schichten wurde dabei an die Morphologie der zu segmentierenden Körperteile angepasst. So konnte zum Beispiel in Bereichen, die sich über mehrere Schichten nur leicht verändern, wie den Beinen des Käfers, größere Abstände zwischen den Schichten gelassen werden, wohingegen in Bereichen hoher Variabilität enger vorsegmentiert werden musste. Im Gegensatz dazu war bei der konventionellen Vorgehensweise eine sehr enge Vorsegmentierung von insgesamt 215 Schichten notwendig, um ein zufriedenstellendes Ergebnis zu erhalten. Hierbei musste jede fünfte Schicht manuell segmentiert werden. Darüber hinaus musste das Ergebnis der anschließenden linearen Interpolation aufwendig manuell nachbearbeitet werden, wohingegen bei der auf Random Walks basierenden Interpolation, bis auf das Entfernen von einigen Ausreißern, keine größeren Korrekturen notwendig waren. Des Weiteren können durch die GPU-basierten Random Walks die üblichen, mit der konventionellen Methode einhergehenden, Interpolationsartefakte vermieden und selbst feine Strukturen, wie beispielsweise Haare, rekonstruiert werden. Insgesamt konnte durch den neuen Ansatz der manuelle Aufwand von insgesamt 77 Stunden auf nur noch 9 Stunden, bei einer gleichzeitigen Qualitätssteigerung, reduziert werden.

Biomedisa benötigt keine Konfiguration von Modellparametern

Das Verfahren zeigt ebenfalls einige wesentliche Vorteile gegenüber den in der Literatur weit verbreiteten und häufig eingesetzten CPU-basierten Verfahren. Eine hohe Segmentierungsgenauigkeit wird bei dem auf Random Walks basierenden Segmentierungsverfahren auch ohne eine komplexe und mühsame Konfiguration unbekannter und in der Regel zu schätzender Modellparameter erreicht (Abschnitt 3.7 & Tabelle 1).
Die Evaluation der Verfahren wurde dabei auf von verschiedenen Experten/-innen manuell segmentierten Schichten durchgeführt. Diese segmentierten Schichten dienten sowohl zur Initialisierung der Verfahren als auch zur Bestimmung der Segmentierungsgenauigkeit. Dabei wurde in einer sogenannten 2-fachen Kreuzvalidierung zuerst jede zweite Schicht zur Initialisierung und die übrigen Schichten zur Evaluation des Segmentierungsergebnisses verwendet. Anschließend wurden die zwei Gruppen vertauscht und die durchschnittliche Genauigkeit bestimmt. Insgesamt wurden somit für die Evaluation nur halb so viele vorsegmentierte Schichten verwendet wie ursprünglich für die Segmentierung der Beispieldatensätze (Abb. 20). Dies erklärt die, insbesondere für sehr große Abstände (z. B. 80 Schichten bei der Theropodenklaue, Tabelle 1), niedrigen Genauigkeiten.
Da es das explizite Ziel dieser Dissertation war, ein einfach zu nutzendes Verfahren zu entwickeln, das unabhängig von der Wahl der Modellparameter bei einer Vielzahl unterschiedlicher Segmentierungsszenarien eine hohe Genauigkeit aufweist, wurden alle Verfahren in ihren Standardkonfigurationen evaluiert. Dabei zeigte sich, dass die Performance der Verfahren GC und RW stark von der Wahl des jeweiligen Modellparameters abhängt. In vielen Fällen wurde mit der Standardkonfiguration dieser Methoden eine sehr niedrige Genauigkeit erzielt. Obwohl diese Parameter für das jeweilige Beispiel angepasst werden können und damit eine etwaige höhere Genauigkeit erzielt werden könnte, erschwert der dadurch entstehende Mehraufwand aufgrund der sehr hohen Rechenzeiten dieser Verfahren eine praktikable Nutzung für sehr große volumetrische Bilddaten.
Das Verfahren GeoS zeigte vor allem Schwierigkeiten bei sehr groß gewählten Abständen zwischen den vorsegmentierten Schichten sowie bei der Rekonstruktion filigraner Strukturen, wie den Haaren des *Trigonopterus* (Abb. 23). Die morphologischen Interpolationen der open-source Software ITK und der kommerziellen Software Amira erzielten wesentlich schlechtere Ergebnisse. Um ein vergleichbares Ergebnis zu erhalten, wird hier eine wesentlich höhere Anzahl vorsegmentierter Schichten benötigt. Außerdem ist bei beiden Methoden keine Extrapolierung über die letzte Schicht hinaus möglich, wodurch jeweils am oberen und unteren Ende der zu segmentierenden Objekte eine zusätzliche, abschließende Schicht segmentiert werden muss.
Ein weiterer Vorteil der GPU-basierten Random Walks ist, dass das als Monte-Carlo-Simulation durchgeführte Verfahren robust gegenüber Fehlern bei der Berechnung ist. Denn einzelne Fehler wie Bit-Flips spielen bei der Gesamtbetrachtung eine eher untergeordnete Rolle. Solche Fehler können technischer Natur sein, z. B. aufgrund immer kleiner und dadurch fehleranfälliger werdenden Transistoren (Tunneleffekt), oder durch äußere Einwirkungen entstehen, z. B. durch elektromagnetische Strahlung. Für die Segmentierung ist es jedoch unerheblich, ob einzelne Random Walks durch eine leichte Störung beeinflusst werden oder nicht.

Hohe Skalierbarkeit der Berechnung

Unter den hier verglichenen Verfahren sind die GPU-basierten Random Walks das einzige Verfahren, das explizit für parallele Computerarchitekturen entwickelt wurde. Durch die sehr hohe Skalierbarkeit der Random Walks lassen sich diese sehr effizient auf GPUs berechnen, wodurch das Verfahren insbesondere für sehr große volumetrische Bilddaten erheblich schneller ist als die mit ihm verglichenen Ansätze, die ebenfalls die Bilddaten berücksichtigen, d. h. RW, GC und GeoS (Abb. 24). Ein schnelles Segmentierungsverfahren ist von sehr großer Bedeutung, da die notwendige Vorsegmentierung gegebenenfalls korrigiert werden muss oder zusätzliche vorsegmentierte Schichten eingefügt werden müssen. Grundsätzlich sind GPU-basierte Implementierungen der hier verglichenen Ansätze (GC, RW und GeoS) möglich. Jedoch ist die Entwicklung und Implementierung effizienter paralleler Lösungen hier ungemein komplexer als die der existierenden CPU-basierten Tools. Des Weiteren hängt die Rechenzeit der Random Walks nicht von der Anzahl der zu segmentierenden Objekte ab, da die Anzahl der insgesamt berechneten Random Walks sich durch eine höhere Anzahl an Label nicht ändert. Die Treffer der jeweiligen Random Walks werden lediglich einer anderen Klasse zugeordnet (Tabelle 2). Das Verfahren wurde ursprünglich für NVIDIA Grafikkarten in Kombination mit CUDA und PyCUDA [Klöckner et al., 2012] entwickelt. Es lässt sich aber ohne Weiteres auf andere Hardwarebeschleuniger, wie FPGAs oder andere Multicore-Prozessoren, übertragen. Die parallele Struktur des Ansatzes führte in der Vergangenheit allein durch die fortschreitende technologische Entwicklung paralleler Hardware zu einer fundamentalen Beschleunigung des Verfahrens, ohne dass dabei eine Veränderung des Quellcodes notwendig war (Abb. 21a). Darüber hinaus ist die Entwicklung parallel arbeitender Grafikprozessoren keineswegs abgeschlossen.

Die GPU-basierten Random Walks sind robust gegenüber Störungen

Zusätzlich ist das GPU-basierte Verfahren sehr robust gegenüber Eingangsstörungen (Abschnitt 3.7.1). Es ist in der Lage, ungenau vorsegmentierte Schichten bis zu einem gewissen Maß zu korrigieren. Dies ist deshalb ein großer Vorteil, da eine inakkurate Vorsegmentierung ein sehr häufig auftretender Fall in der dreidimensionalen Bildverarbeitung ist. Die Anpassung ist möglich, da die Vorsegmentierungen nicht als unveränderlich angenommen werden, sondern ausschließlich der Initialisierung der Random Walks dienen. Erst die berechnete Verteilung der Random Walks bestimmt die endgültige Segmentierung, und dabei auch im Bereich der vorsegmentierten Schichten. Alle anderen Verfahren hingegen betrachten die Vorsegmentierung als unveränderliche Randbedingung. Dies hat jedoch unmittelbar zur Folge, dass eine fehlerhafte Initialisierung nicht nur fehlerhaft bleibt, sondern auch die Konvergenz der Lösung dieser Verfahren sehr stark beeinflusst (Abb. 25).

Die schnelle Segmentierung erleichtert eine interaktive Vorgehensweise

Im Allgemeinen ist es ratsam ein Biomedisa-Projekt mit einer niedrigen, von der Morphologie des zu segmentierenden Objektes abhängigen, Anzahl vorsegmentierter Schichten zu starten und dann sukzessive zusätzliche Schichten dort hinzuzufügen, wo die Segmentierung ein mangelhaftes Ergebnis aufweist. Diese Vorgehensweise wird dadurch begünstigt, dass die kurze Verarbeitungszeit bei einer kleiner werdenden Anzahl vorsegmentierter Schichten weiter reduziert wird, wohingegen die Verarbeitung der verglichenen Verfahren mit einer sinkenden Anzahl mehr Zeit in Anspruch nimmt (Tabelle 2). Das interaktive Hinzufügen von vorsegmentierten Schichten ist wesentlich effizienter als eine Vorsegmentierung vieler Schichten, ohne zu wissen, wie viele Schichten tatsächlich notwendig sind. Beispielsweise genügten für den *Trigonopterus* Rüsselkäfer insgesamt 37 Schichten für eine Genauigkeit der Segmentierung von 97,80 %. Eine Vorsegmentierung von wesentlich mehr Schichten (108 (98,14 %) und 54 (98,00 %)) führte jedoch nicht zu einer dem wesentlich größeren Aufwand gerechtfertigten Verbesserung. Präzise formuliert: Das Verdoppeln der vorsegmentierten Schichten von 54 auf 108 verdoppelte den Arbeitsaufwand von 13 auf 26 Stunden, wobei lediglich eine Steigerung der Genauigkeit von 0,14 % erzielt wurde. Wohingegen die Wahl von 27 Schichten (97,28 %) erste deutliche Fehler im Ergebnis aufweist. Noch weniger vorsegmentierte Schichten resultieren in einer stark fehlerbehafteten Segmentierung (93,10 % bei einer Vorsegmentierung von jeder 80. Schichte und 81,37 % bei jeder 160. Schicht), jedoch wurden selbst hier noch große Teile des Käfers korrekt segmentiert (Abb. 19).

Künstliche neuronale Netze für eine automatisierte Segmentierung

Die Ergebnisse der halbautomatischen Segmentierung können als Trainingsdaten für ein tiefes künstliches neuronales Netz verwendet werden, um somit eine vollautomatische Segmentierung einer großen Anzahl von Proben mit ähnlichen Strukturen, wie bspw. dem Gehirn von Insekten, zu ermöglichen (Kapitel 4). Auf diese Weise werden groß angelegte vergleichende quantitative Analysen ermöglicht, um beispielsweise Erkenntnisse über den Zusammenhang zwischen Verhalten und kognitiver Anpassung im Gehirn von Honigbienen zu gewinnen.

Universalität der Biomedisa Online-Plattform

Sowohl das semi-automatische als auch das automatische Verfahren wurden in die ebenfalls entwickelte, frei verfügbare und leicht zu nutzende Online-Anwendung Biomedisa integriert (Kapitel 5). Neben den GPU-basierten Random Walks und den künstlichen neuronalen Netzen enthält die Online-Plattform viele weitere Features zur Nachbearbeitung und Visualisierung der Ergebnisse (Abschnitt 5.2). Darüber hinaus lässt sich die Unsicherheit, mit der das Ergebnis erzielt wurde, darstellen (Abschnitt 3.6). Die Lokalisierung von Un-

sicherheiten kann dabei helfen, schlecht vorsegmentierte Schichten oder Regionen mit zu wenig Referenzschichten zu identifizieren, um diese anschließend zu korrigieren oder weitere Schichten hinzuzufügen. Aufgrund der Modularität der Plattform lässt sich diese leicht um weitere Funktionalitäten erweitern (Abschnitt 5.3.3). Durch die Unterstützung aller gängigen Datenformate lässt sich die Anwendung problemlos mit einer anderen Segmentierungssoftware kombinieren. Dies ermöglicht dem/der Nutzer/-in, seinen bereits etablierten Workflow beizubehalten, gleichzeitig aber den Arbeitsschritt der Segmentierung signifikant zu beschleunigen. Das Verfahren und die Anwendung wurden ursprünglich für CT und MRT Daten entwickelt, sie sind jedoch ohne Weiteres auf andere volumetrische Bilddaten wie die konfokale Laser-Scanning- und Focused-Ion-Beam-Mikroskopie oder auf die Analyse histologischer Schnitte übertragbar. Obwohl die Entwicklung durch Anwendungen in der Biologie und Medizin motiviert wurde, lässt sich diese auch zur Analyse von Daten anderer Disziplinen wie der Geologie und Materialwissenschaften nutzen.

Ausblick & Weiterentwicklungen

Die automatische Segmentierung mit künstlichen neuronalen Netzen weist nach wie vor erhebliche Einschränkungen auf. Es werden sehr große Mengen an Trainingsdaten benötigt, die in der Regel aufwendig manuell oder semi-automatisch erzeugt werden müssen. Sowohl die Erstellung der Trainingsdaten als auch das Training der Netze sind bis heute nicht nur äußerst arbeits- und zeitaufwendig, sondern auch sehr fehleranfällig. Die Benutzer/-innen können an zahlreichen Stellen viele Fehler machen und insbesondere ist häufig keine oder nur eine schlechte Interaktion des/der Benutzers/-in mit der Software möglich. Die Ergebnisse müssen in der Regel im Nachhinein aufwendig manuell korrigiert und das Netzwerk danach von Grund auf neu trainiert werden. Die Handhabung der künstlichen neuronalen Netze ist insgesamt nach wie vor äußerst komplex. Bereits trainierte Netze lassen sich nicht ohne Weiteres auf veränderte Konfigurationen der Bildaufnahme an den Instrumenten oder der Datenvorverarbeitung und auf neue Problemstellungen und Probentypen übertragen, d. h. die Segmentierung der Künstlichen Intelligenz (KI) scheitert immer dann, wenn die Daten zu stark von den Trainingsdaten abweichen, sei es aufgrund einer abweichenden Morphologie (Abb. 33), anderen CT-Bildaufnahme-, Bearbeitungs-, oder Rekonstruktionsmodalitäten, oder weil ganz neue Probentypen segmentiert werden sollen. Dabei sind die Grenzen der jeweiligen Anwendbarkeit der trainierten Netze schwer vorhersagbar. Im Ergebnis werden Arbeitszeit und technische Ressourcen aus wissenschaftlicher, wirtschaftlicher und energetischer Sicht nicht effizient genutzt. Die erzielten morphologischen Ergebnisse sind häufig zu ungenau und viele Anwendungen praktisch zu aufwendig. Dadurch sind morphologische Studien mit hohen Anforderungen an die statistische Signifikanz und an die zu untersuchende Probenzahl in der Regel nur mit Einschränkungen möglich.

Herausforderungen: Eine erste Herausforderung ist daher die substantielle Reduktion des Arbeitsaufwandes zur Erstellung der Trainingsdaten. Die zweite besteht darin, trotz der hohen Komplexität der Arbeit mit künstlichen neuronalen Netzen, eine schnelle Lösung

für die Anpassung der CNNs an neuartige Szenarien und diverse Bilddaten zu ermöglichen. Drittens muss der Aufwand der Fehlerbehandlung durch eine interaktive Fehlerkorrektur reduziert werden. Insgesamt erfordert eine schnelle und leichtere Anpassung der Modelle bzw. CNNs (insb. auch an neue Serien) eine intuitiv verständliche und ressourcenschonende Interaktion mit dem/der Nutzer/-in.

Interaktives Lernen mit Biomedisa: Mit Biomedisa können für eine KI-basierte Segmentierung tiefe CNNs auf bereits vollständig segmentierten volumetrischen Daten trainiert werden (Abschnitt 4.4.1). Die Ergebnisse sind jedoch häufig fehlerbehaftet und müssen manuell nachbearbeitet werden. Die Fehler entstehen, da die Trainingsdaten in der Regel nicht alle Möglichkeiten abdecken können (Abb. 33). Um die Vielfalt der Trainingsdaten zu erhöhen, werden aktuell die korrigierten Ergebnisse den Trainingsdaten hinzugefügt und das CNN erneut trainiert (Abschnitt 5.1). Dies ist jedoch umständlich und das erneute Training benötigt viel Zeit (bei 26 volumetrischen Trainingsdaten von nur $256 \times 256 \times 256$ Voxel benötigt Biomedisa unter Verwendung von 4 NVIDIA Tesla V100 etwa 13 Stunden) und somit erhebliche Ressourcen. Für ein effizientes Ressourcenmanagement und eine dazu notwendige nutzerfreundlichere Handhabung soll in Biomedisa eine KI-basierte interaktive Segmentierung ermöglicht werden. Dabei werden im Segmentierungsergebnis durch wenige Klicks die fehlerhaften Stellen markiert und dem richtigen Objekt zugeordnet. Jede Korrektur durch den/die Benutzer/-in führt automatisch zu einer Aktualisierung des Modells. Da die Anpassung des gesamten künstlichen neuronalen Netzes sehr rechenintensiv und zeitaufwendig ist, sollen bestehende Techniken evaluiert und weiterentwickelt werden, bei denen durch Einführung von Hilfsvariablen der Backpropagation-Algorithmus (Abschnitt 4.2) nur noch auf einen Teil des Netzwerks angewandt werden muss, wie f-BRS (*Feature Backpropagating Refinement Scheme* [Kontogianni et al., 2020]). Dies ermöglicht die Aktualisierung des Modells im laufenden Betrieb. Für die Interaktion mit dem/der Benutzer/-in soll in Biomedisa eine visuelle Schnittstelle integriert werden, die es ermöglicht, die Klicks online durchzuführen. Durch die interaktive Segmentierung lässt sich auf die bereits gelernten Strukturen und Formen eines fertig trainierten Netzes aufbauen und so beispielsweise auch der Aufwand für neue ähnliche Datensätze reduzieren. Zum Beispiel bei einem Transfer des auf Bienengehirnen trainierten CNNs (Abschnitt 4.4) zu anderen Insekten.

Optimales Experimentelles Design (Active Learning): Die manuelle Segmentierung zur Erzeugung von Trainingsdaten ist äußerst aufwendig. Typischerweise werden einzelne Schichten eines Volumens manuell oder das ganze Volumen semi-automatisch segmentiert und anhand dieser Referenzdaten ein CNN trainiert. Um den Aufwand möglichst gering zu halten und nicht unnötige, weil redundante, Schichten manuell zu segmentieren, lautet die Frage: Welches zusätzlich segmentierte 3D-Volumen oder welche Schicht im Volumen liefert den größtmöglichen Informationsgewinn? Ziel muss daher die Identifizierung der Schichten sein, die im Sinne eines geeigneten Optimalitätskriteriums bestmöglich zum Training des künstlichen neuronalen Netzes beitragen. Bei diesem aktiven Lernen werden dem/der Benutzer/-in interaktiv die Schichten, Regionen oder ganze 3D-Bilder vorgeschlagen, die den höchsten zusätzlichen Nutzen generieren, um somit den Gesamtaufwand auf ein Mini-

mum zu reduzieren. Darüber hinaus können auf Basis der Wahrscheinlichkeit, mit der ein künstliches neuronales Netz seine Ergebnisse erzielt, bereits mehrere Optionen zur Auswahl gestellt werden, aus denen der/die Benutzer/-in nur noch die richtige Segmentierung auswählen muss. Auf diese Weise können in kürzester Zeit zusätzliche Trainingsdaten generiert und das künstliche neuronale Netz sukzessive optimiert werden.

Selbstüberwachtes Lernen zur Identifikation relevanter 3D-Strukturen: Durch die geeignete Wahl einer Metrik können künstliche neuronale Netze selbstüberwacht 3D-Strukturen, Texturen und Formen identifizieren, die für die vorsegmentierten Objekte in einem Datensatz charakteristisch sind. Derzeit verwendet Biomedisa ausschließlich die Differenz und Varianz von Grauwerten in einer lokalen Umgebung, um dadurch die Gewichte der Random Walks zu bestimmen (Abschnitt 3.1). Das selbstüberwacht trainierte Netzwerk hingegen erzeugt für jeden Bildausschnitt einen Merkmalsvektor, der die jeweilige Textur repräsentiert. Dadurch können die Random Walks z. B. Faser-, Zell-, Körnerstrukturen oder andere komplexe Texturen für die Segmentierung berücksichtigen (d. h. komplexe semantische Deskriptoren aus den Daten selbst extrahieren), ohne dass diese explizit dem Algorithmus mitgeteilt werden müssen. Insbesondere lassen sich auf diese Weise auch diffuse Objektkanten, durch die die klassischen Random Walks hindurchlaufen, zur Segmentierung identifizieren. Als Ergebnis wird der Verlauf der Random Walks durch ähnliche Strukturen beliebiger Komplexität und somit deren Segmentierung ermöglicht.

KI-gestützte effiziente Ressourcennutzung: Künstliche neuronale Netze können aufwendige numerische Simulationen ersetzen. Dabei dienen Informationen über das Simulationsgebiet, wie Objekteigenschaften und Randbedingungen, zusammen mit den Simulationsergebnissen als Trainingsgrundlage [Zakeri et al., 2020]. Die semi-automatische Segmentierung von Biomedisa basiert auf der Durchführung vieler gewichteter Random Walks, die in vorsegmentierten Schichten starten. Anhand der Verteilung der Random Walks lässt sich das 3D-Volumen in verschiedene Objekte unterteilen. Die Segmentierung mit Random Walks ist für sehr große Bilddaten zeitaufwendig (Abschnitt 5.4) und benötigt den Einsatz vieler Hardwareressourcen (GPUs). Auf der Biomedisa Online-Plattform wurden mit dieser Methode bereits eine große Anzahl unterschiedlicher Daten segmentiert - täglich kommen weitere hinzu (Abb. 44). Die verfügbaren Bilddaten können zusammen mit den vorsegmentierten Schichten und den durch die Random Walks erzeugten Ergebnissen dazu verwendet werden, um ein CNN zu trainieren, das die bereits sehr erfolgreiche Segmentierung durch Random Walks imitiert. Diese Simulation der Random Walks könnte um mehrere Größenordnungen schneller erfolgen als durch deren explizite Berechnung. Da die automatische Segmentierung durch das bereits trainierte Netzwerk auch auf einer CPU erfolgen kann, würde ein auf diese Weise trainiertes Netzwerk die semi-automatische Segmentierung durch sehr wenige Ressourcen, beispielsweise auf einem Notebook, und eine weitere Skalierung der Online-Plattform erlauben.

Literatur

[Aerts et al., 2014] Aerts, H. J. W. L., Velazquez, E. R., Leijenaar, R. T. H., Parmar, C., Grossmann, P., Carvalho, S., Bussink, J., Monshouwer, R., Haibe-Kains, B., Rietveld, D., Hoebers, F., Rietbergen, M. M., Leemans, C. R., Dekker, A., Quackenbush, J., Gillies, R. J., and Lambin, P. (2014). Decoding tumour phenotype by noninvasive imaging using a quantitative radiomics approach. *Nature Communications*, 5(1):4006.

[Akkus et al., 2017] Akkus, Z., Galimzianova, A., Hoogi, A., Rubin, D. L., and Erickson, B. J. (2017). Deep Learning for Brain MRI Segmentation: State of the Art and Future Directions. *Journal of Digital Imaging*, 30(4):449–459.

[Arganda-Carreras et al., 2017] Arganda-Carreras, I., Kaynig, V., Rueden, C., Eliceiri, K. W., Schindelin, J., Cardona, A., and Sebastian Seung, H. (2017). Trainable Weka Segmentation: a machine learning tool for microscopy pixel classification. *Bioinformatics*, 33(15):2424–2426.

[Bai and Sapiro, 2007] Bai, X. and Sapiro, G. (2007). A Geodesic Framework for Fast Interactive Image and Video Segmentation and Matting. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8, Rio de Janeiro, Brazil. IEEE.

[Baid et al., 2021] Baid, U., Ghodasara, S., Mohan, S., Bilello, M., Calabrese, E., Colak, E., Farahani, K., Kalpathy-Cramer, J., Kitamura, F. C., Pati, S., Prevedello, L. M., Rudie, J. D., Sako, C., Shinohara, R. T., Bergquist, T., Chai, R., Eddy, J., Elliott, J., Reade, W., Schaffter, T., Yu, T., Zheng, J., Moawad, A. W., Coelho, L. O., McDonnell, O., Miller, E., Moron, F. E., Oswood, M. C., Shih, R. Y., Siakallis, L., Bronstein, Y., Mason, J. R., Miller, A. F., Choudhary, G., Agarwal, A., Besada, C. H., Derakhshan, J. J., Diogo, M. C., Do-Dai, D. D., Farage, L., Go, J. L., Hadi, M., Hill, V. B., Iv, M., Joyner, D., Lincoln, C., Lotan, E., Miyakoshi, A., Sanchez-Montano, M., Nath, J., Nguyen, X. V., Nicolas-Jilwan, M., Jimenez, J. O., Ozturk, K., Petrovic, B. D., Shah, C., Shah, L. M., Sharma, M., Simsek, O., Singh, A. K., Soman, S., Statsevych, V., Weinberg, B. D., Young, R. J., Ikuta, I., Agarwal, A. K., Cambron, S. C., Silbergleit, R., Dusoi, A., Postma, A. A., Letourneau-Guillon, L., Perez-Carrillo, G. J. G., Saha, A., Soni, N., Zaharchuk, G., Zohrabian, V. M., Chen, Y., Cekic, M. M., Rahman, A., Small, J. E., Sethi, V., Davatzikos, C., Mongan, J., Hess, C., Cha, S., Villanueva-Meyer, J., Freymann, J. B., Kirby, J. S., Wiestler, B., Crivellaro, P., Colen, R. R., Kotrotsou, A., Marcus, D., Milchenko, M., Nazeri, A., Fathallah-Shaykh, H., Wiest, R., Jakab, A., Weber, M.-A., Mahajan, A., Menze, B., Flanders, A. E., and Bakas, S. (2021). The RSNA-ASNR-MICCAI BraTS 2021 Benchmark on Brain Tumor Segmentation and Radiogenomic Classification. *arXiv:2107.02314 [cs]*. arXiv: 2107.02314.

[Bakas et al., 2017] Bakas, S., Akbari, H., Sotiras, A., Bilello, M., Rozycki, M., Kirby, J. S., Freymann, J. B., Farahani, K., and Davatzikos, C. (2017). Advancing The Cancer Genome Atlas glioma MRI collections with expert segmentation labels and radiomic features. *Scientific Data*, 4(1):170117.

[Balanta-Melo et al., 2019] Balanta-Melo, J., Bemmann, M., Ibacache, V. T., Kupczik, K., and Buvinic, S. (2019). Three-dimensional assessment of enamel and dentine in mouse molar teeth during masseter muscle hypofunction. *Revista Estomatología*, 26(2):30–37.

[Balanta-Melo et al., 2018] Balanta-Melo, J., Torres-Quintana, M. A., Bemmann, M., Vega, C., González, C., Kupczik, K., Toro-Ibacache, V., and Buvinic, S. (2018). Masseter muscle atrophy impairs bone quality of the mandibular condyle but not the alveolar process early after induction. *Journal of Oral Rehabilitation*, page joor.12747.

[Barron, 1993] Barron, A. (1993). Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945.

[Belevich et al., 2016] Belevich, I., Joensuu, M., Kumar, D., Vihinen, H., and Jokitalo, E. (2016). Microscopy Image Browser: A Platform for Segmentation and Analysis of Multidimensional Datasets. *PLOS Biology*, 14(1):e1002340.

[Bemmann et al., 2021] Bemmann, M., Schulz-Kornas, E., Hammel, J. U., Hipp, A., Moosmann, J., Herrel, A., Rack, A., Radespiel, U., Zimmermann, E., Kaiser, T. M., and Kupczik, K. (2021). Movement analysis of primate molar teeth under load using synchrotron X-ray microtomography. *Journal of Structural Biology*, 213(1):107658.

[Berg et al., 2019] Berg, S., Kutra, D., Kroeger, T., Straehle, C. N., Kausler, B. X., Haubold, C., Schiegg, M., Ales, J., Beier, T., Rudy, M., Eren, K., Cervantes, J. I., Xu, B., Beuttenmueller, F., Wolny, A., Zhang, C., Koethe, U., Hamprecht, F. A., and Kreshuk, A. (2019). ilastik: interactive machine learning for (bio)image analysis. *Nature Methods*, 16(12):1226–1232.

[Boudinot et al., 2021] Boudinot, B. E., Moosdorf, O. T. D., Beutel, R. G., and Richter, A. (2021). Anatomy and evolution of the head of *Dorylus helvolus* (Formicidae: Dorylinae): Patterns of sex- and caste-limited traits in the sausagefly and the driver ant. *Journal of Morphology*, 282(11):1616–1658.

[Boykov and Funka-Lea, 2006] Boykov, Y. and Funka-Lea, G. (2006). Graph Cuts and Efficient N-D Image Segmentation. *International Journal of Computer Vision*, 70(2):109–131.

[Boykov and Jolly, 2001] Boykov, Y. and Jolly, M.-P. (2001). Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 1, pages 105–112, Vancouver, BC, Canada. IEEE Comput. Soc.

[Brandt et al., 2005] Brandt, R., Rohlfing, T., Rybak, J., Krofczik, S., Maye, A., Westerhoff, M., Hege, H.-C., and Menzel, R. (2005). Three-dimensional average-shape atlas of the honeybee brain and its applications. *The Journal of Comparative Neurology*, 492(1):1–19.

[Buduma and Locascio, 2017] Buduma, N. and Locascio, N. (2017). *Fundamentals of deep learning: designing next-generation machine intelligence algorithms*. O'Reilly Media, Sebastopol, CA, first edition edition. OCLC: ocn992798385.

[Buvinic et al., 2021] Buvinic, S., Balanta-Melo, J., Kupczik, K., Vásquez, W., Beato, C., and Toro-Ibacache, V. (2021). Muscle-Bone Crosstalk in the Masticatory System: From Biomechanical to Molecular Interactions. *Frontiers in Endocrinology*, 11.

[Bär, 2010] Bär, C. (2010). *Elementare Differentialgeometrie*. De Gruyter, Berlin, 2 edition.

[Bölcskei et al., 2019] Bölcskei, H., Grohs, P., Kutyniok, G., and Petersen, P. (2019). Optimal Approximation with Sparsely Connected Deep Neural Networks. *SIAM Journal on Mathematics of Data Science*, 1(1):8–45.

[Chan and Vese, 2001] Chan, T. and Vese, L. (2001). Active contours without edges. *IEEE Transactions on Image Processing*, 10(2):266–277.

[Cheng et al., 2014] Cheng, J., Grossman, M., and McKercher, T. (2014). *Professional Cuda® C Programming*. Wrox programmer to programmer. Wrox/Wiley, Indianapolis, Ind.

[Chenyang Xu and Prince, 1998] Chenyang Xu and Prince, J. (1998). Snakes, shapes, and gradient vector flow. *IEEE Transactions on Image Processing*, 7(3):359–369.

[Christ et al., 2016] Christ, P. F., Elshaer, M. E. A., Ettlinger, F., Tatavarty, S., Bickel, M., Bilic, P., Rempfler, M., Armbruster, M., Hofmann, F., D'Anastasi, M., Sommer, W. H., Ahmadi, S.-A., and Menze, B. H. (2016). Automatic Liver and Lesion Segmentation in CT Using Cascaded Fully Convolutional Neural Networks and 3D Conditional Random Fields. In Ourselin, S., Joskowicz, L., Sabuncu, M. R., Unal, G., and Wells, W., editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016*, volume 9901, pages 415–423. Springer International Publishing, Cham. Series Title: Lecture Notes in Computer Science.

[Criminisi et al., 2008] Criminisi, A., Sharp, T., and Blake, A. (2008). GeoS: Geodesic Image Segmentation. In Forsyth, D., Torr, P., and Zisserman, A., editors, *Computer Vision – ECCV 2008*, volume 5302, pages 99–112. Springer Berlin Heidelberg, Berlin, Heidelberg. Series Title: Lecture Notes in Computer Science.

[Csader et al., 2021] Csader, M., Mayer, K., Betz, O., Fischer, S., and Eggs, B. (2021). Ovipositor of the braconid wasp *Habrobracon hebetor*: structural and functional aspects. *Journal of Hymenoptera Research*, 83:73–99.

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314.

[de Bakker et al., 2016] de Bakker, B. S., de Jong, K. H., Hagoort, J., de Bree, K., Besselink, C. T., de Kanter, F. E. C., Veldhuis, T., Bais, B., Schildmeijer, R., Ruijter, J. M., Oostra, R.-J., Christoffels, V. M., and Moorman, A. F. M. (2016). An interactive three-dimensional digital atlas and quantitative database of human development. *Science*, 354(6315):aag0053.

[dos Santos Rolo et al., 2014] dos Santos Rolo, T., Ershov, A., van de Kamp, T., and Baumbach, T. (2014). In vivo X-ray cine-tomography for tracking morphological dynamics. *Proceedings of the National Academy of Sciences*, 111(11):3921–3926.

[Dumbravă et al., 2016] Dumbravă, M. D., Rothschild, B. M., Weishampel, D. B., Csiki-Sava, Z., Andrei, R. A., Acheson, K. A., and Codrea, V. A. (2016). A dinosaurian facial deformity and the first occurrence of ameloblastoma in the fossil record. *Scientific Reports*, 6(1):29271.

[Dunbar, 1998] Dunbar, R. I. M. (1998). The social brain hypothesis. *Evolutionary Anthropology: Issues, News, and Reviews*, 6(5):178–190.

[Eldan and Shamir, 2016] Eldan, R. and Shamir, O. (2016). The Power of Depth for Feedforward Neural Networks. *arXiv:1512.03965 [cs, stat]*. arXiv: 1512.03965.

[Engelkes, 2020] Engelkes, K. (2020). Accuracy of bone segmentation and surface generation strategies analyzed by using synthetic CT volumes. *Journal of Anatomy*.

[Evans, 2013] Evans, L. C. (2013). *An introduction to stochastic differential equations*. American Mathematical Society, Providence, Rhode Island.

[Evans and Spruck, 1991] Evans, L. C. and Spruck, J. (1991). Motion of level sets by mean curvature. I. *Journal of Differential Geometry*, 33(3).

[Fahrbach et al., 1998] Fahrbach, S. E., Moore, D., Capaldi, E. A., Farris, S. M., and Robinson, G. E. (1998). Experience-expectant plasticity in the mushroom bodies of the honeybee. *Learning & Memory (Cold Spring Harbor, N.Y.)*, 5(1-2):115–123.

[Ford and Fulkerson, 1956] Ford, L. R. and Fulkerson, D. R. (1956). Maximal Flow Through a Network. *Canadian Journal of Mathematics*, 8:399–404.

[Giurfa, 2013] Giurfa, M. (2013). Cognition with few neurons: higher-order learning in insects. *Trends in Neurosciences*, 36(5):285–294.

[Grady, 2006] Grady, L. (2006). Random Walks for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11):1768–1783.

[Gross et al., 2019] Gross, V., Müller, M., Hehn, L., Ferstl, S., Allner, S., Dierolf, M., Achterhold, K., Mayer, G., and Pfeiffer, F. (2019). X-ray imaging of a water bear offers a new look at tardigrade internal anatomy. *Zoological Letters*, 5(1):14.

[Gutiérrez et al., 2020] Gutiérrez, Y., Ott, D., and Scherber, C. (2020). Direct and indirect effects of plant diversity and phenoxy herbicide application on the development and reproduction of a polyphagous herbivore. *Scientific Reports*, 10(1).

[Göttler et al., 2021] Göttler, C., Amador, G., van de Kamp, T., Zuber, M., Böhler, L., Siegwart, R., and Sitti, M. (2021). Fluid mechanics and rheology of the jumping spider body fluid. *Soft Matter*, 17(22):5532–5539.

[Harvati et al., 2019] Harvati, K., Röding, C., Bosman, A. M., Karakostis, F. A., Grün, R., Stringer, C., Karkanas, P., Thompson, N. C., Koutoulidis, V., Moulopoulos, L. A., Gorgoulis, V. G., and Kouloukoussa, M. (2019). Apidima Cave fossils provide earliest evidence of *Homo sapiens* in Eurasia. *Nature*, 571(7766):500–504.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Las Vegas, NV, USA. IEEE.

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

[Jandt et al., 2014] Jandt, J. M., Bengston, S., Pinter-Wollman, N., Pruitt, J. N., Raine, N. E., Dornhaus, A., and Sih, A. (2014). Behavioural syndromes and social insects: personality at multiple levels: Behavioural syndromes and social insects. *Biological Reviews*, 89(1):48–67.

[Jeremiah, 2011] Jeremiah (2011). CUDA Thread Execution Model. https://www.3dgep.com/cuda-thread-execution-model/#more-1913. Besucht: 04.11.2021.

[Jones et al., 2019] Jones, M. E. H., Button, D. J., Barrett, P. M., and Porro, L. B. (2019). Digital dissection of the head of the rock dove (*Columba livia*) using contrast-enhanced computed tomography. *Zoological Letters*, 5(1):17.

[Kallinowski et al., 2021] Kallinowski, F., Ludwig, Y., Gutjahr, D., Gerhard, C., Schulte-Hörmann, H., Krimmel, L., Lesch, C., Uhr, K., Lösel, P., Voß, S., Heuveline, V., Vollmer, M., Görich, J., and Nessel, R. (2021). Biomechanical Influences on Mesh-Related Complications in Incisional Hernia Repair. *Frontiers in Surgery*, 8:763957.

[Kikinis et al., 2014] Kikinis, R., Pieper, S. D., and Vosburgh, K. G. (2014). 3D Slicer: A Platform for Subject-Specific Image Analysis, Visualization, and Clinical Support. In Jolesz, F. A., editor, *Intraoperative Imaging and Image-Guided Therapy*, pages 277–289. Springer New York, New York, NY.

[Klöckner et al., 2012] Klöckner, A., Pinto, N., Lee, Y., Catanzaro, B., Ivanov, P., and Fasih, A. (2012). PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation. *Parallel Computing*, 38(3):157–174.

[Kontogianni et al., 2020] Kontogianni, T., Gygli, M., Uijlings, J., and Ferrari, V. (2020). Continuous Adaptation for Interactive Object Segmentation by Learning from Corrections. *arXiv:1911.12709 [cs]*. arXiv: 1911.12709.

[Lecun et al., 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[L'Ecuyer, 1999] L'Ecuyer, P. (1999). Good Parameters and Implementations for Combined Multiple Recursive Random Number Generators. *Operations Research*, 47(1):159–164.

[Leshno et al., 1993] Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867.

[Litjens et al., 2017] Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., van der Laak, J. A., van Ginneken, B., and Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42:60–88.

[Lorensen and Cline, 1987] Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques - SIGGRAPH '87*, pages 163–169, Not Known. ACM Press.

[Luders et al., 2009] Luders, E., Narr, K. L., Thompson, P. M., and Toga, A. W. (2009). Neuroanatomical correlates of intelligence. *Intelligence*, 37(2):156–163.

[Lösel and Heuveline, 2016] Lösel, P. and Heuveline, V. (2016). Enhancing a diffusion algorithm for 4D image segmentation using local information. *Proc. SPIE*, 9784:97842L.

[Lösel and Heuveline, 2017] Lösel, P. and Heuveline, V. (2017). A GPU Based Diffusion Method for Whole-Heart and Great Vessel Segmentation. In Zuluaga, M. A., Bhatia, K., Kainz, B., Moghari, M. H., and Pace, D. F., editors, *Reconstruction, Segmentation, and Analysis of Medical Images*, volume 10129, pages 121–128. Springer International Publishing, Cham. Series Title: Lecture Notes in Computer Science.

[Lösel et al., Vorb] Lösel, P. D., Monchanin, C., Lebrun, R., Jayme, A., Relle, J., Devaud, J.-M., Heuveline, V., and Lihoreau, M. (Vorb). Inter-individual variability in honey bee brain size and organisation revealed by micro-CT imaging and deep learning. *Manuskript in Vorbereitung.*

[Lösel et al., 2020] Lösel, P. D., van de Kamp, T., Jayme, A., Ershov, A., Faragó, T., Pichler, O., Tan Jerome, N., Aadepu, N., Bremer, S., Chilingaryan, S. A., Heethoff, M., Kopmann, A., Odar, J., Schmelzle, S., Zuber, M., Wittbrodt, J., Baumbach, T., and Heuveline, V. (2020). Introducing Biomedisa as an open-source online platform for biomedical image segmentation. *Nature Communications*, 11(1):5577.

[Maier-Hein et al., 2018] Maier-Hein, L., Eisenmann, M., Reinke, A., Onogur, S., Stankovic, M., Scholz, P., Arbel, T., Bogunovic, H., Bradley, A. P., Carass, A., Feldmann, C., Frangi, A. F., Full, P. M., van Ginneken, B., Hanbury, A., Honauer, K., Kozubek, M., Landman, B. A., März, K., Maier, O., Maier-Hein, K., Menze, B. H., Müller, H., Neher, P. F., Niessen, W., Rajpoot, N., Sharp, G. C., Sirinukunwattana, K., Speidel, S., Stock, C., Stoyanov, D., Taha, A. A., van der Sommen, F., Wang, C.-W., Weber, M.-A., Zheng, G., Jannin, P., and Kopp-Schneider, A. (2018). Why rankings of biomedical image analysis competitions should be interpreted with care. *Nature Communications*, 9(1):5217.

[Maire and Withers, 2014] Maire, E. and Withers, P. J. (2014). Quantitative X-ray tomography. *International Materials Reviews*, 59(1):1–43.

[Marquez-Neila et al., 2014] Marquez-Neila, P., Baumela, L., and Alvarez, L. (2014). A Morphological Approach to Curvature-Based Evolution of Curves and Surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(1):2–17.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133.

[Menze et al., 2015] Menze, B. H., Jakab, A., Bauer, S., Kalpathy-Cramer, J., Farahani, K., Kirby, J., Burren, Y., Porz, N., Slotboom, J., Wiest, R., Lanczi, L., Gerstner, E., Weber, M.-A., Arbel, T., Avants, B. B., Ayache, N., Buendia, P., Collins, D. L., Cordier, N., Corso, J. J., Criminisi, A., Das, T., Delingette, H., Demiralp, C., Durst, C. R., Dojat, M., Doyle, S., Festa, J., Forbes, F., Geremia, E., Glocker, B., Golland, P., Guo, X., Hamamci, A., Iftekharuddin, K. M., Jena, R., John, N. M., Konukoglu, E., Lashkari, D., Mariz, J. A., Meier, R., Pereira, S., Precup, D., Price, S. J., Raviv, T. R., Reza, S. M. S., Ryan, M., Sarikaya, D., Schwartz, L., Shin, H.-C., Shotton, J., Silva, C. A., Sousa, N., Subbanna, N. K., Szekely, G., Taylor, T. J., Thomas, O. M., Tustison, N. J., Unal, G., Vasseur, F., Wintermark, M., Ye, D. H., Zhao, L., Zhao, B., Zikic, D., Prastawa, M., Reyes, M., and Van Leemput, K. (2015). The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS). *IEEE Transactions on Medical Imaging*, 34(10):1993–2024.

[Menzel, 2012] Menzel, R. (2012). The honeybee as a model for understanding the basis of cognition. *Nature Reviews Neuroscience*, 13(11):758–768.

[Mhaskar, 1993] Mhaskar, H. N. (1993). Approximation properties of a multilayered feedforward artificial neural network. *Advances in Computational Mathematics*, 1(1):61–80.

[Mikó et al., 2018a] Mikó, I., Trietsch, C., van de Kamp, T., Masner, L., Ulmer, J. M., Yoder, M. J., Zuber, M., Sandall, E. L., Baumbach, T., and Deans, A. R. (2018a). Revision of *Trassedia* (Hymenoptera: Ceraphronidae), an Evolutionary Relict With an Unusual Distribution. *Insect Systematics and Diversity*, 2(6).

[Mikó et al., 2018b] Mikó, I., van de Kamp, T., Trietsch, C., Ulmer, J. M., Zuber, M., Baumbach, T., and Deans, A. R. (2018b). A new megaspilid wasp from Eocene Baltic amber (Hymenoptera: Ceraphronoidea), with notes on two non-ceraphronoid families: Radiophronidae and Stigmaphronidae. *PeerJ*, 6:e5174.

[Moser et al., 2021] Moser, M., Burks, R. A., Ulmer, J. M., Heraty, J. M., van de Kamp, T., and Krogmann, L. (2021). Taxonomic description and phylogenetic placement of two new species of *Spalangiopelta* (Hymenoptera: Pteromalidae: Ceinae) from Eocene Baltic amber. *PeerJ*, 9:e10939.

[NVIDIA, 2021] NVIDIA (2021). CUDA C++ Programming Guide. https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf. Besucht: 04.11.2021.

[Osher and Sethian, 1988] Osher, S. and Sethian, J. A. (1988). Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79(1):12–49.

[Oswald, 1990] Oswald, P. (1990). On the degree of nonlinear spline approximation in Besov-Sobolev spaces. *Journal of Approximation Theory*, 61(2):131–157.

[Pace et al., 2015] Pace, D. F., Dalca, A. V., Geva, T., Powell, A. J., Moghari, M. H., and Golland, P. (2015). Interactive Whole-Heart Segmentation in Congenital Heart Disease. In Navab, N., Hornegger, J., Wells, W. M., and Frangi, A. F., editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, volume 9351, pages 80–88. Springer International Publishing, Cham. Series Title: Lecture Notes in Computer Science.

[Panser et al., 2016] Panser, K., Tirian, L., Schulze, F., Villalba, S., Jefferis, G., Bühler, K., and Straw, A. (2016). Automatic Segmentation of *Drosophila* Neural Compartments Using GAL4 Expression Data Reveals Novel Visual Pathways. *Current Biology*, 26(15):1943–1954.

[Pardo et al., 2017] Pardo, J. D., Szostakiwskyj, M., Ahlberg, P. E., and Anderson, J. S. (2017). Hidden morphological diversity among early tetrapods. *Nature*, 546(7660):642–645.

[Patterson et al., 1988] Patterson, D. A., Gibson, G., and Katz, R. H. (1988). A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the 1988 ACM SIGMOD international conference on Management of data - SIGMOD '88*, pages 109–116, Chicago, Illinois, United States. ACM Press.

[Petersen and Voigtlaender, 2018] Petersen, P. and Voigtlaender, F. (2018). Optimal approximation of piecewise smooth functions using deep ReLU neural networks. *Neural Networks*, 108:296–330.

[Petersen, 2020] Petersen, P. C. (2020). Neural Network Theory. http://www.pc-petersen.eu/Neural_Network_Theory.pdf. Besucht: 04.11.2021.

[Pietschnig et al., 2015] Pietschnig, J., Penke, L., Wicherts, J. M., Zeiler, M., and Voracek, M. (2015). Meta-analysis of associations between human brain volume and intelligence differences: How strong are they and what do they mean? *Neuroscience & Biobehavioral Reviews*, 57:411–432.

[Pinkus, 1999] Pinkus, A. (1999). Approximation theory of the MLP model in neural networks. *Acta Numerica*, 8:143–195.

[Qvarnström et al., 2021] Qvarnström, M., Fikáček, M., Vikberg Wernström, J., Huld, S., Beutel, R. G., Arriaga-Varela, E., Ahlberg, P. E., and Niedźwiedzki, G. (2021). Exceptionally preserved beetles in a Triassic coprolite of putative dinosauriform origin. *Current Biology*, page S0960982221006746.

[Richard and Lippmann, 1991] Richard, M. D. and Lippmann, R. P. (1991). Neural Network Classifiers Estimate Bayesian *a posteriori* Probabilities. *Neural Computation*, 3(4):461–483.

[Richter et al., 2021a] Richter, A., Garcia, F. H., Keller, R. A., Billen, J., Katzke, J., Boudinot, B. E., Economo, E. P., and Beutel, R. G. (2021a). The head anatomy of *Protanilla lini* (Hymenoptera: Formicidae: Leptanillinae), with a hypothesis of their mandibular movement. *Myrmecological News*.

[Richter et al., 2020] Richter, A., Hita Garcia, F., Keller, R. A., Billen, J., Economo, E. P., and Beutel, R. G. (2020). Comparative analysis of worker head anatomy of *Formica* and *Brachyponera* (Hymenoptera: Formicidae). *Arthropod Systematics & Phylogeny*.

[Richter et al., 2021b] Richter, A., Schoeters, E., and Billen, J. (2021b). Morphology and closing mechanism of the mandibular gland orifice in ants (Hymenoptera: Formicidae). *Journal of Morphology*.

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In Navab, N., Hornegger, J., Wells, W. M., and Frangi, A. F., editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, volume 9351, pages 234–241. Springer International Publishing, Cham.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

[Rother et al., 2004] Rother, C., Kolmogorov, V., and Blake, A. (2004). "GrabCut": interactive foreground extraction using iterated graph cuts. In *ACM SIGGRAPH 2004 Papers on - SIGGRAPH '04*, page 309, Los Angeles, California. ACM Press.

[Schindelin et al., 2012] Schindelin, J., Arganda-Carreras, I., Frise, E., Kaynig, V., Longair, M., Pietzsch, T., Preibisch, S., Rueden, C., Saalfeld, S., Schmid, B., Tinevez, J.-Y., White, D. J., Hartenstein, V., Eliceiri, K., Tomancak, P., and Cardona, A. (2012). Fiji: an open-source platform for biological-image analysis. *Nature Methods*, 9(7):676–682.

[Schmelzle et al., 2017] Schmelzle, S., van de Kamp, T., Heethoff, M., Heuveline, V., Lösel, P., Becker, J., Beckmann, F., Schluenzen, F., Hammel, J. U., Kopmann, A., Mexner, W., Vogelgesang, M., Jerome, N. T., Betz, O., Beutel, R., Wipfler, B., Blanke, A., Harzsch, S., Hörnig, M., and Baumbach, T. (2017). The NOVA project: maximizing beam time efficiency through synergistic analyses of SRμCT data. *Proc. SPIE*, 10391:103910P.

[Sethian, 2015] Sethian, J. A. (2015). Fast Marching Methods. In Engquist, B., editor, *Encyclopedia of Applied and Computational Mathematics*, pages 490–493. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Shaham et al., 2018] Shaham, U., Cloninger, A., and Coifman, R. R. (2018). Provable approximation properties for deep neural networks. *Applied and Computational Harmonic Analysis*, 44(3):537–557.

[Sigal et al., 2018] Sigal, Y. M., Zhou, R., and Zhuang, X. (2018). Visualizing and discovering cellular structures with super-resolution microscopy. *Science*, 361(6405):880–887.

[Smith et al., 2020] Smith, D. B., Arce, A. N., Ramos Rodrigues, A., Bischoff, P. H., Burris, D., Ahmed, F., and Gill, R. J. (2020). Insecticide exposure during brood or early-adult development reduces brain growth and impairs adult learning in bumblebees. *Proceedings of the Royal Society B: Biological Sciences*, 287(1922):20192442.

[Smith et al., 2016] Smith, D. B., Bernhardt, G., Raine, N. E., Abel, R. L., Sykes, D., Ahmed, F., Pedroso, I., and Gill, R. J. (2016). Exploring miniature insect brains using micro-CT scanning techniques. *Scientific Reports*, 6(1):21768.

[Sombke et al., 2015] Sombke, A., Lipke, E., Michalik, P., Uhl, G., and Harzsch, S. (2015). Potential and limitations of X-Ray micro-computed tomography in arthropod neuroanatomy: A methodological and comparative survey. *Journal of Comparative Neurology*, 523(8):1281–1295.

[Stegmaier et al., 2016] Stegmaier, J., Amat, F., Lemon, W., McDole, K., Wan, Y., Teodoro, G., Mikut, R., and Keller, P. (2016). Real-Time Three-Dimensional Cell Segmentation in Large-Scale Microscopy Data of Developing Embryos. *Developmental Cell*, 36(2):225–240.

[Telgarsky, 2016] Telgarsky, M. (2016). Benefits of depth in neural networks. *arXiv:1602.04485 [cs, stat]*. arXiv: 1602.04485.

[Toivanen, 1996] Toivanen, P. J. (1996). New geodosic distance transforms for gray-scale images. *Pattern Recognition Letters*, 17(5):437–450.

[van de Kamp et al., 2018] van de Kamp, T., Schwermann, A. H., dos Santos Rolo, T., Lösel, P. D., Engler, T., Etter, W., Faragó, T., Göttlicher, J., Heuveline, V., Kopmann, A., Mähler, B., Mörs, T., Odar, J., Rust, J., Tan Jerome, N., Vogelgesang, M., Baumbach, T., and Krogmann, L. (2018). Parasitoid biology preserved in mineralized fossils. *Nature Communications*, 9(1).

[van de Kamp et al., 2011] van de Kamp, T., Vagovic, P., Baumbach, T., and Riedel, A. (2011). A Biological Screw in a Beetle's Leg. *Science*, 333(6038):52–52.

[van der Walt et al., 2014] van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., and Yu, T. (2014). scikit-image: image processing in Python. *PeerJ*, 2:e453.

[Vese and Chan, 2002] Vese, L. A. and Chan, T. F. (2002). A Multiphase Level Set Framework for Image Segmentation Using the Mumford and Shah Model. *International Journal of Computer Vision*, 50(3):271–293.

[Vogelgesang et al., 2016] Vogelgesang, M., Farago, T., Morgeneyer, T. F., Helfen, L., dos Santos Rolo, T., Myagotin, A., and Baumbach, T. (2016). Real-time image-content-based beamline control for smart 4D X-ray imaging. *Journal of Synchrotron Radiation*, 23(5):1254–1263.

[Wallner et al., 2018] Wallner, J., Hochegger, K., Chen, X., Mischak, I., Reinbacher, K., Pau, M., Zrnc, T., Schwenzer-Zimmerer, K., Zemann, W., Schmalstieg, D., and Egger, J. (2018). Clinical evaluation of semi-automatic open-source algorithmic software segmentation of the mandibular bone: Practical feasibility and assessment of a new course of action. *PLOS ONE*, 13(5):e0196378.

[Wallner et al., 2019] Wallner, J., Mischak, I., and Jan Egger (2019). Computed tomography data collection of the complete human mandible and valid clinical ground truth models. *Scientific Data*, 6(1):190003.

[Wang et al., 2016] Wang, G., Zuluaga, M. A., Pratt, R., Aertsen, M., Doel, T., Klusmann, M., David, A. L., Deprest, J., Vercauteren, T., and Ourselin, S. (2016). Slic-Seg: A minimally interactive segmentation of the placenta from sparse and motion-corrupted fetal MRI in multiple views. *Medical Image Analysis*, 34:137–147.

[Wang et al., 2021] Wang, X., Gao, S., Wang, M., and Duan, Z. (2021). A marching cube algorithm based on edge growth. *Virtual Reality & Intelligent Hardware*, 3(4):336–349.

[Weinhardt et al., 2018] Weinhardt, V., Shkarin, R., Wernet, T., Wittbrodt, J., Baumbach, T., and Loosli, F. (2018). Quantitative morphometric analysis of adult teleost fish by X-ray computed tomography. *Scientific Reports*, 8(1):16531.

[Withers et al., 1993] Withers, G. S., Fahrbach, S. E., and Robinson, G. E. (1993). Selective neuroanatomical plasticity and division of labour in the honeybee. *Nature*, 364(6434):238–240.

[Wolf et al., 2005] Wolf, I., Vetter, M., Wegner, I., Böttger, T., Nolden, M., Schöbinger, M., Hastenteufel, M., Kunert, T., and Meinzer, H.-P. (2005). The Medical Imaging Interaction Toolkit. *Medical Image Analysis*, 9(6):594–604.

[Yarotsky, 2017] Yarotsky, D. (2017). Error bounds for approximations with deep ReLU networks. *Neural Networks*, 94:103–114.

[Yatziv et al., 2006] Yatziv, L., Bartesaghi, A., and Sapiro, G. (2006). O(N) implementation of the fast marching algorithm. *Journal of Computational Physics*, 212(2):393–399.

[Yushkevich et al., 2006] Yushkevich, P. A., Piven, J., Hazlett, H. C., Smith, R. G., Ho, S., Gee, J. C., and Gerig, G. (2006). User-guided 3D active contour segmentation of anatomical structures: Significantly improved efficiency and reliability. *NeuroImage*, 31(3):1116–1128.

[Zakeri et al., 2020] Zakeri, B., Monsefi, A. K., and Darafarin, B. (2020). Deep Learning Prediction of Heat Propagation on 2-D Domain via Numerical Solution. In Bohlouli, M., Sadeghi Bigham, B., Narimani, Z., Vasighi, M., and Ansari, E., editors, *Data Science: From Research to Application*, volume 45, pages 161–174. Springer International Publishing, Cham. Series Title: Lecture Notes on Data Engineering and Communications Technologies.

[Zhou, 2018] Zhou, D.-X. (2018). Deep distributed convolutional neural networks: Universality. *Analysis and Applications*, 16(06):895–919.

[Zhou, 2020] Zhou, D.-X. (2020). Universality of deep convolutional neural networks. *Applied and Computational Harmonic Analysis*, 48(2):787–794.