Universität Bremen

Deep Anatomy

Universität Bremen

Fachbereich 03 - Mathematik / Informatik

Projektbericht

DeepAnatomy

Projektbetreuer:	Prof. Dr.-Ing. Horst K. Hahn		`horst.hahn@mevis.fraunhofer.de`	
	Dr. Hans Meine		`hans.meine@mevis.fraunhofer.de`	
	Felix Thielke		`felix.thielke@mevis.fraunhofer.de`	
Semester:	WiSe 2023/24			
	SoSe 2024			
Autoren:	Tamari Bayer	`tambayer`	Emrullah Baykara	`ebaykara`
	Felix Drees	`fdrees`	Lukas Garbade	`lgarbade`
	Arbresh Gashi	`arbresh1`	Hanno Henke	`hanno1`
	Ardit Keta	`aketa`	Jonathan Kinkel	`joki1`
	Hannah Köper	`hkoeper`	Tobias Liese	`tobias5`
	Ole Mahlstädt	`olma`	Semjon Nirmann	`semnir`
	Jorma Reiners	`b_wh73ed`	Jessica Repty	`jrepty`
	Freja Sender	`freja`	Jörn Steffens	`steffjoe`
	Aziz Tas	`aziz1`	Tom Wolff	`tomwolff`

Email-Adressen jeweils `@uni-bremen.de`

Inhaltsverzeichnis

1 Einleitung

Jorma, Freja, Semjon

DeepAnatomy ist ein jährlich stattfindendes Projekt, das abwechselnd als Bachelor- und Masterprojekt vom Institut für Digitale Medizin - Fraunhofer MEVIS angeboten wird. Die Teilnehmer*innen können dabei flexibel Themen aus dem Bereich des Deep Learning auswählen und bearbeiten, in der Regel unter dem Deckmantel einer übergeordneten Thematik. Gleichermaßen besteht die Möglichkeit, bestehende Software die im Rahmen der Forschung verwendet wird, auszubauen oder weiterzuentwickeln. Dabei kann entweder auf vorherigen Projekten aufgebaut werden, oder aber es wird ein neues Thema gewählt, so wie in diesem Jahr.

Im Fokus des diesjährigen Projekts stand das Oberthema "Sarkopenie" – Muskelschwund. Das Ziel bestand darin, Muskelschwund mithilfe neuronaler Netze zu erkennen sowie die dafür verwendete Infrastruktur auszubauen. Ein zusätzlicher Schwerpunkt lag auf der Vor- und Nachbearbeitung der Daten.

Innerhalb dieses Oberthemas konnten wir uns den verschiedensten Teilbereichen der Softwareentwicklung widmen:

- **Infrastruktur** Hier sind insbesondere die im Abschnitt 3 Vorbereitungen sowie zum Teil auch in 4.2 Cluster-Dashboard enthaltenen Kapitel interessant: 3.1 Keras Migration, 3.2 SPLAT, 3.3 TrainingLoop, sowie 3.5 GitLab URLs und 4.2.2 Blossom.

- **Frontend Entwicklung** Die Projekte 4.2.1 Logger und 4.2.2 Blossom beschäftigten sich vor allem mit Frontend-Arbeiten zur Überwachung eines laufenden Trainings.

- **Deep Learning** Im Projekt 4.1 Sarkopenie wurde ein Netz auf die Erkennung der Muskelmasse trainiert und die Ergebnisse mittels Postprocessing weiterverarbeitet.

- **Modellevaluation** In dem Projekt 5 Modellevaluation: Muskelsegmentierung wurden drei verschiedene, bestehende, Modelle zur Erkennung der Muskelmasse und das in 4.1 Sarkopenie erarbeitete einander gegenübergestellt und verglichen.

Neben der Motivation, die verwendeten Werkzeuge, wie z.B. MevisLab, das Cluster-Dashboard und Blossom, zu ergänzen, sollten Teile der Ergebnisse nach Abschluss des Projekts in ein zukünftiges Forschungsprojekt in Zusammenarbeit mit dem Universitätsklinikum Hamburg-Eppendorf einfließen.

2 Projektmanagement

Jorma, Semjon

2.1 Organisation

Für die Organisation des Projekts wurde ein wöchentlich stattfindendes Meeting festgelegt. Diese Treffen dienten der gemeinsamen Besprechung von Fortschritten und aktuellen Ereignissen sowie der Selbstorganisation der Projektteilnehmer in Arbeitsgruppen. Parallel dazu erfolgte eine Betreuung durch die Mentoren Dr. Hans Meine und Felix Thielke, die richtungsweisende Unterstützung und Beratung bei auftretenden Problemen boten. Es gab vor Allem zu Beginn des Projekts regelmäßig Vorträge der Mentoren, um die notwendigen theoretischen Grundlagen für die Teilprojekte zu vermitteln. Zusätzlich ergänzten die Projektteilnehmer in eigenen Vorträgen Wissen zu relevanten Themengebieten und Werkzeugen.
Über das Semester hinweg fanden wiederholt Feedbackgespräche statt, in denen die Teilnehmer sowohl Selbst- als auch gegenseitige Einschätzungen abgaben und damit die Möglichkeit boten, die eigene Leistung für das Projekt zu evaluieren und die Arbeitsweise zu verbessern.

2.2 Werkzeuge

Für die Koordination und Bearbeitung der Projekte kamen verschiedene Werkzeuge zum Einsatz. Das Fraunhofer MEVIS-Institut verwendet intern eine Vielzahl an Werkzeugen und Anwendungen, welche uns zur Verfügung gestellt wurden.

Confluence wurde verwendet, um Ideen, Fortschritte und die zugrundliegende Motivation zu dokumentiere. Dort wurden zentrale Informationen gespeichert und Termine koordiniert. Zudem ermöglichte Confluence das Verwalten von Protokollen und das Sammeln und Organisieren von relevanten Ressourcen.

MevisLab ist eine Anwendung vom Fraunhofer MEVIS, die für die Verarbeitung und Analyse medizinischer Bilddaten entwickelt wurde. MevisLab bietet eine Umgebung, in der Nutzer visuell mit komplexer Bildverarbeitung arbeiten können.

Mattermost ist eine Open-Source-Plattform für den Nachrichtenaustausch. Es wurde als primäres Tool für die Projektkommunikation genutzt.

SATORI ist eine browserbasierte Webanwendung zum ansehen und annotieren medizinischer Bilddaten. Darüber hinaus kann SATORI auch mit Plugins um einige Funktionen erweitert werden, zum Beispiel Upload von DICOM-Daten direkt im Browser oder Starten von Trainings und Presegmentationen.

Gaia ist ein CIFS/NFS basiertes Netzlaufwerk, welches von MEVIS bereitgestellt wurde. Verbundene Nutzer*innen können darüber Dateien Teilen.

HashiCorp Nomad ist ein Anwendung zum Orchestrieren von Containern. Anwendungen laufen im Cloud-basierten Umfeld von Fraunhofer MEVIS in Docker-Containern auf leistungsfähigen Server-Rechnern. Die Zuweisung der Ressourcen zu sogenannten „Jobs“ findet mit Hilfe von Nomad statt.

Das **Cluster-Dashboard** ist eine Webanwendung, welche die Erstellung und das Starten eines Deep Learning Trainings auf dem Nomad-Cluster ermöglicht. In diesem Rahmen werden unter anderem Instanzen von MevisLab zur Konfiguration und Vor- bzw. Nachbearbeitung von Daten sowie auch der RedLeaf Anwendung auf dem Cluster alloziert und gestartet. Dort werden ebenfalls Pfade zu den auf Gaia liegenden Ressourcen angegeben, welche für die Konfiguration des Trainings nötig sind.

Blossom ist eine Anwendung für die Echtzeit-Visualisierung von Trainingsmetriken, wie zum Beispiel dem Trainingsverlust. Zu Begin des Projekts war die Anwendung nicht mehr funktionstüchtig beziehungsweise aktiv.

RedLeaf ist ein Framework, das unter der Verwendung von Pytorch und Keras die Ausführung von Deep Learning Trainings ermöglicht.

GitLab wurde als zentraler Dienst zur Versionsverwaltung in der Softwareentwicklung verwendet.

Neben den vorgestellten Hauptwerkzeugen kamen noch eine Vielzahl weiterer Tools wie der Barracuda-VPN, die Daten-Management Plattform Girder und Software-Frameworks wie Quasar und Vue.js zum Einsatz.

3 Vorbereitungen

Freja

Die im Folgenden beschriebenen Projekte haben sich mit der Zurverfügungstellung grundlegender Infrastruktur, wie dem Update des Keras Deep Learning Frameworks von Version 2 auf Version 3 beschäftigt. Weiterhin geht es um Infrastruktur zum Annotieren der Daten (SPLAT, TrainingLoop).

3.1 Keras Migration

Hanno, Jonathan

Keras ist eine in Python geschriebene Deep Learning Bibliothek, welche die Implementierung von neuronalen Netzen ermöglicht. Im Herbst 2023 wurde die neuere Version Keras 3 veröffentlicht. Diese Version realisiert zusätzlich zu dem Keras 2 kompatiblem Backend Framework *TensorFlow* die Kompatibilität mit den Frameworks *JAX* und *PyTorch*. Kompatibilität bedeutet hier, dass jedes Keras-Modell ohne Änderungen am Quellcode dynamisch auf jedem dieser Frameworks ausgeführt werden kann. Dafür wurden die in Keras 2 verwendeten TensorFlow-Funktionen als Keras-Funktionen mit äquivalenter Funktionalität bereitgestellt. Das Multi-Backend erwirkt den Vorteil, dass für jedes Modell das Backend ausgewählt werden kann, welches hinsichtlich eines bestimmten Kriteriums wie Effizienz oder Performanz die besten Ergebnisse erzielt. Zudem kann jedes Modell alle Vorteile der unterschiedlichen Frameworks nutzen.

Implementierung Die Erkenntnisse aus ersten Auseinandersetzungen mit der Softwarearchitektur von MEVIS ergab, dass Keras größtenteils über *TensorFlow* als Deeplearning API zum Training von Modellen verwendet wurde. Für die Gewährleistung einer einheitlichen Softwareumgebung, die für mehrere Entwickler nutzbar ist, wird *Docker* verwendet. Als Vorbereitung für die Versionsmigration musste dementsprechend ein neues Docker Image gebaut werden, welches die neuere Version von Keras beinhaltet. Für die Sicherstellung einer erfolgreichen und korrekten Migration verfügt MEVIS über weitreichende Tests. Es wurden im Laufe des Projekts zwei verschiedene Arten von Tests durchgeführt. Zum einen sogenannte rld-pytest (RedLeaf Docker pytests), welche die jeweiligen Dateien einzeln testen und den *integration tests*, in denen Trainings auf dem Cluster von MEVIS durchgeführt werden.

Um die oben genannten Vorteile der Verwendung eines variablen Backends zu nutzen, musste die Implementierung von Keras nicht über *TensorFlow*, sondern über Keras selbst erfolgen. Die oben bereits erwähnten TensorFlow-Funktionen wurden in die jeweils äquivalenten Keras-Funktionen überführt. Für diesen Schritt sind die `import` Aufrufe der Form `from tensorflow import keras` und `from tensorflow.keras import xyz` in Aufrufe der Form `import keras` sowie `from keras import xyz` umgeändert worden. Außerdem mussten Aufrufe innerhalb der Klassen wie `tf.keras.*` durch `keras.*` ersetzt werden. Für die meisten Funktionen wurde die Bezeichnung einfach übernommen, daher war dieser Schritt sehr simpel. Von einigen Funktionen wurde die Bezeichnung jedoch überarbeitet, weshalb diese Änderungen manuell angepasst wurden.

Außerdem gab es weitere Funktionen aus der alten Keras Version, die zwar Teil des Keras repository waren, aber nicht zur offiziellen API gehörten. Deswegen gab es das Problem, dass einige dieser Funktionen in der neuen Version entweder umbenannt oder entfernt wurden. Da ihre Nutzung nicht offiziell empfohlen wird, gibt es hierzu keine Dokumentationen von Keras selbst. Infolgedessen mussten die Funktionen angepasst und in den meisten Fällen durch eine eigene Implementierung ersetzt werden.

Nach den ersten Durchführungen der rld-pytests stellte sich heraus, dass einige der verwendeten Bibliotheken und Frameworks nicht mit der neuen Version von Keras

kompatibel waren. Insbesondere waren tensorflowprobability und ein ONNX-Modul problematisch. Mit der Veröffentlichung von tensorflow 2.16 im Frühjahr 2024 wurde das erste Problem behoben. Das zweite Problem wurde gelöst indem das benötigte Modul in angepasster Version aus einem individuellen Repository heraus importiert wurde. Nach beheben dieser Fehler liefen die Pytests zum ersten Mal erfolgreich an.

Im Anschluss daran wurden erstmals Integrationstests durchgeführt. Dies erforderte erneut das Erstellen einiger Dockerimages, um die richtige Konfiguration der Umgebung sicherzustellen. Die Auswertung der Integrationstests ergab, dass die auftretenden Fehler mit den Fehlern der rld-pytests übereinstimmten, weshalb der Fokus der Fehlerbehebung wieder zurück auf die rld-pytests gelegt wurde.

Ausblick und Fazit Das Migrieren von Versionen, sowie das Sicherstellen von Versionskompatibilitäten sind komplexe und zeitaufwendige Arbeiten. Dies trifft insbesondere zu wenn es sich um eine große Codebasis mit vielen zusammenhängenden Komponenten handelt, wie es im Fraunhofer MEVIS der Fall ist. Das Erstellen neuer Dockerimages, sowie die Nutzung eines eigenen Git Branches, ermöglicht es allen Entwickler*Innen von MEVIS mit der gleichen Arbeitsumgebung arbeiten zu können. Dies führt dazu, dass alle aktuellen und zukünftigen Mitarbeiter*Innen die Arbeit fortführen und fertigstellen können. Das Erreichen des Anlaufens der rld-pytests war ein Meilenstein in Bezug auf die Migration. Auch die Erkenntnis der Problematik, dass nicht zur API gehörige Funktionen von Keras innerhalb der Codebasis von MEVIS verwendet wurden, wird der Stablität des Codes zukünftig zugutekommen. Etliche auftretende Fehler innerhalb der Tests konnten gefixt werden. Allerdings ist die Migration noch nicht fertiggestellt. Eine Vorhersage des restlichen Arbeitsaufwandes ist nicht zu treffen, da sich immer wieder neue Fehler durch das Lösen anderer Fehler ergeben. Grundsätzlich konnten im Laufe des Projekts immer wieder Erfolge verbucht werden. Auch wenn die Fertigstellung der Migration noch nicht beendet ist, wurden hier die wichtigsten Grundbausteine gelegt, um diese Arbeit vervollständigen zu können.

3.2 SPLAT

Tamari

Das übergeordnete Ziel des Projekts war, wie in dem Abschnitt 1 Einleitung erläutert, sowohl die Konfiguration und das Training, als auch die Evaluation eines Neuronalen Netzes, sodass zwei Strukturen, Bauch- und Rückenmuskel im unteren Bereich der Wirbelsäule (L2-L4), erkannt werden können. Für diese Schritte sind die Daten, die dem Netz zur Verfügung gestellt werden, essentiell. Diese präsentierten sich in Form von Computer Tomographie (CT) Scans. Die Daten selbst stammen von einer vergangenen KiTS-Challenge (Kidney Tumor Segmentation Challenge).

Gegeben waren die Daten von 59 Patient*innen, deren CT-Scans jeweils unterschiedliche Anzahlen an Schichten beinhaltet haben. Die Scans werden in drei disjunkte Mengen - *training*, *test* und *validation* - aufgeteilt. Die Daten aus der Gruppe *training* fließen in dem Training mit ein. Sie werden mittels Preprocessing (vgl. Abschnitt 4.1.1 Pre-Processing) präpariert und dem Netz als Lernmaterial gegeben. Das Training läuft viele Iterationen durch. Je n Iterationen wird das Netz evaluiert und die Ergebnisse dieser Evaluierung werden anschließend zur Auswahl der besten Parameter für das trainierte Netz verwendet. Hierfür werden die Daten aus der Gruppe *validation* verwendet.

Die Gruppe *test* dient zur Evaluierung des trainierten Netzes. Dieser Gruppe zugeordnete CT-Scans werden als Input für das Netz verwendet. Den Output stellen entsprechend die segmentierten CT-Scans dar.

Für die drei Gruppen reichen jedoch nur die reinen CT-Scans nicht aus. Die CT-Scans bieten einen genauen Einblick in den Körper, wobei Organe, Muskeln, Knochen, Fett und weitere Bestandteile in unterschiedlichen Grauwerten (Hounsfield units) dargestellt werden. Das Netz verwendet u.a. diese Werte, um oben aufgezählte Strukturen voneinander abzugrenzen. Das Training eines Netzes verlangt die Annotationen einiger Schichten der CT Scans. Dies ermöglicht dem Algorithmus eigenständig Muster, Strukturen und räumliche Anordnungen zu erkennen und diese den annotierten Labels zuzuordnen. Die Annotationen wurden entsprechend, getrennt für Bauch- und Rückenmuskeln, von den Mitwirkenden

dieses Projekts händisch eingezeichnet. Die Annotationen dienen zudem als Referenz bei der Evaluierung des Netzes. Hier werden die Ergebnisse des Netzes den Annotationen gegenübergestellt und unterschiedliche Metriken ausgerechnet (vgl. Abschnitt 5 Modellevaluation: Muskelsegmentierung).

Im Rahmen dieses Unterprojekts wurde das Annotationstool von CSI (Contouring. Snapping. Interpolation) auf SPLAT (Spase Label Annotation Tool) umgestellt. Die medizinische Bildgebung war für viele Mitwirkende dieses Projekts eine Herausforderung, wodurch die Annotationen aufgrund des mangelhaften Wissens nicht die benötigte Genauigkeit aufgewiesen haben. Für diesen Use-Case hat sich die flexible Natur von SPLAT herauskristallisiert, wodurch die Umstellung anschließend motiviert wurde.

CSI ist sowohl ein Makromodul im MeVisLab, mit dem sich die Annotationen der CT-Scans realisieren lassen. Es setzt sich u.a. aus den Untermodulen für das Contouring, Snapping und Interpolation zusammen und implementiert einen universellen Editor für die Erstellung und Bearbeitung von Konturen. Der Fokus liegt auf die Verwendung von Freiformkonturen, um eine präzise Konturierung zu ermöglichen. Das Untermodul für das Snapping zielt darauf ab, die Genauigkeit der Konturen durch automatische Anpassung an Objektgrenzen zu verbessern. Mit dem Interpolationsmodul können unregelmäßig abgetastete Konturpunkte ergänzt werden. Damit trägt das Untermodul zur Glättung und Kontinuität der Konturen bei.

Mit CSI annotierte Schichten sind binär - es gibt eine binäre Klassifizierung von Voxeln. Sie sind entweder dem Vorder- oder dem Hintergrund zugeordnet. Die Zuordnung wird anhand der Annotationen bestimmt, wobei CSI nur die explizite Annotation von dem Vordergrund erlaubt. Während des Trainings werden alle Voxel einer Schicht berücksichtigt.

Ähnlich zu CSI ist SPLAT eine Methode zur Kennzeichnung von Bildregionen in der medizinischen Bildverarbeitung. Im Gegensatz zu CSI ermöglicht SPLAT eine differenziertere Markierung von Bildbereichen. Mit SPLAT können nicht nur Hintergrund und Vordergrund gekennzeichnet werden, sondern es können auch zusätzliche Kategorien wie undefinierte oder unsichere Bereiche markiert werden.

Dies ermöglicht eine präzisere Charakterisierung von Bildregionen, insbesondere in komplexen medizinischen Bildern, in denen die Zuordnung von Pixeln zu bestimmten Strukturen oder Geweben schwierig sein kann.

Die Umstellung erforderte sowohl die Anpassung der Konfigurationen von SATORI, welches zum Annotieren der CT-Schichten dient, als auch die Preprocessing-Schritte, die die Trainingdaten aufbereiten.

3.3 TrainingLoop

Ole, Tamari

Das Modul TrainingLoop ist eine Erweiterung für das browserbasierte Annotations-Tool SATORI. Es ermöglicht, neben den Annotationen auch weitere Schritte des Workflows abzudecken. Dadurch kann ein schnellerer und einheitlicher Arbeitsablauf ermöglicht werden, ohne weitere Anwendungen wie MeVisLab hinzu ziehen zu müssen.

Stattdessen kann mit TrainingLoop aus SATORI heraus ein Training eines Modells, zum Beispiel mit gerade neu annotierten Datensätzen, gestartet werden. Außerdem können mit trainierten Modellen auch Presegmentationen durchgeführt werden. Deren Ergebnisse können ebenfalls in dem Viewer von SATORI angezeigt werden.

Ziel im Laufe des Projekts war es, TrainingLoop für unser Sarkopenie-Setup mit SPLAT verwenden zu können. Zusätzlich sollte noch eine Quality-of-Life Verbesserung implementiert werden, die es ermöglichen soll, direkt nach einem abgeschlossenen Training Presegmentationen mit dem neuen Modell zu starten. Zuvor musste abgewartet und aktiv vom Nutzer geguckt werden, ob ein Trainingsdurchlauf bereits abgeschlossen ist, um danach Presegmentationen mit dem neu trainierten Modell zu starten, um die Ergebnisse davon zu prüfen.

Presegmentation Für die Presegmentierung wird ein trainiertes Modell benötigt. Außerdem müssen die erkannten Strukturen entsprechend der Konfiguration im Sarkopnie-Setup benannt sein. Zu beachten ist auch, dass die eigentliche Arbeit der Presegmentation als eigener Job auf dem Cluster gestartet wird. Dies unterscheidet sich zu einem lokalen SATORI Setup, bei dem auf einem lokalem Rechner das Training in einem Docker-Container gestartet wird. Stattdessen gibt es einen `PresegmentationManager`, welcher permanent auf dem Cluster läuft, neue Aufträge entgegen nimmt, eine entsprechende Work-Queue pflegt und, wenn Arbeit in dieser Work-Queue ansteht, `PresegmentationWorker` startet. Dieser `PresegmentationWorker` läuft auch auf dem Cluster, hat aber zusätzlich GPU-Ressourcen, welche die anderen Jobs nicht benötigen. Somit werden nicht permanent, sondern nur bei tatsächlich anstehender Arbeit, GPU-Ressourcen reserviert. Für den Manager und den Worker gibt es jeweils eine Job-Definitions-Datei (`.hcl`) die entsprechend dem Sarkopenie-Setup angepasst werden musste.

Die Presegmentation kann aus dem TrainingLoop heraus gestartet werden. Über einen dedizierten Tab bekommen die Anwender der SATORI die Möglichkeit, beliebige Fälle mit einem aktiven Modell zu trainieren. Die Presegmentation selbst wird mittels Konfigurationsdateien für SATORI bestimmt. Dafür wird eine `.mlab`-Datei zusammen mit dazugehörigem Skript gefordert. Diese Dateien nehmen die Daten aus SATORI entgegen und erzeugen daraus die Presegmentation-Strukturen. Nachdem die Fälle und das Modell ausgewählt wurden, werden sie an dem Skript bzw. an der `.mlab`-Datei weitergegeben. Die `.mlab`-Datei stellt sicher, dass die CT-Scans das richtige Format für eine möglichst gute Segmentierung haben. Außerdem enthält diese Datei Module, die zur Segmentierung nötig sind. Insgesamt sorgt die `.mlab`-Datei für die richtige Ein- und Ausgabe für die Presegmentation. Das Skript dient zum Konfigurieren der anzulegenden Strukturen und stößt die tatsächliche Segmentierung an. Der Einstiegspunkt ist an dieser Stelle die `update`-Funktion, welche ausgeführt wird, wenn eine Presegmentation ansteht. Dabei werden die Daten (CT-Scans eines Falls) als Eingabe für die Module verwendet und schrittweise die Korrektheit der Ausgabe geprüft. Anschließend werden die Scans segmentiert und die daraus entstandene Strukturen gespeichert.

Die Presegmentation-Strukturen werden gesondert in SATORI dargestellt, damit die Anwender sie als solche erkennen können. Dies ist ein sehr nützliches Tool für die Verbesserung der Annotationen, da man hiermit die Schwächen eines Modells erkennen und proaktiv beheben kann.

Automatische Presegmentation nach einem Training Sollte über neue Callbacks in `TrainingLoopTrainingCallbacks.py` gestartet werden. Durch verschiedene technische Probleme haben wir jedoch am Ende des Projekts keine lokalen Tests davon durchführen können. Darunter fallen zum Beispiel Fehler mit Grafikkarten-Treibern, MeVisLab-Lizensierung im lokal gestarteten Data-Preparation Container, `perUserConfig` für die `TrainingLoopExpert` Option in der `extensions.json` die nicht immer greift.

3.4 Upload-Logger

Jorma

Motivation Das Folgeprojekt in Zusammenarbeit mit der Universitätsklinik Hamburg-Eppendorf untersucht Sarkopenie anhand medizinischer Daten. Dabei ist geplant, DICOM-Dateien auf eine SATORI-Instanz hochladen zu können. Laut gesetzlichen Vorgaben muss hierfür dokumentiert werden, wer die Daten hochgeladen hat und wann dies geschehen ist. Für SATORI existiert bereits ein Modul, welches das Hochladen und Löschen von DICOM-Dateien ermöglicht. Allerdings fehlt bisher eine Funktion, welche den Zeitpunkt und den Urheber des Uploads gemäß den gesetzlichen Richtlinien protokolliert.

Methode Für die Entwicklung wurde eine lokale Instanz von SATORI verwendet. Das bestehende DICOM-Upload-Modul wurde um einen Konfigurationsparameter erweitert, der es ermöglicht, das Logging zu de- und aktivieren. Es wurden verschiedene Varianten geprüft, um festzustellen, welche Daten dokumentiert werden sollten und in welchem Format dies geschieht. Neben dem Upload werden ebenfalls weitere Metadaten der DICOM-Dateien aufgezeichnet.

2024-04-07T23:56:07.265	USER A	UPLOAD	1.3.6.1.4.1.34261.18356146217557.7204.1716155766.0	1.2.826.0.1.3680043.8.1055.1.20111102150758591.92402465.76095170	1.33563947522054E+038
2024-05-08T23:56:15.366	USER B	DELETE	1.3.6.1.4.1.34261.18356146217557.7204.1716155766.0	1.2.826.0.1.3680043.8.1055.1.20111102150758591.92402465.76095170	1.33563947522054E+038
2024-05-11T23:56:32.082	USER B	UPLOAD	1.4.3.7.1.3.29661.31356146217327.4102.1710356791.0	1.3.763.1.6.3639265.8.1055.1.827436503270434 83.90324753.12370659	1.24354672346264E+028

Abbildung 2: Beispiele protokollierter Daten

Ergebnis Wie in der Abbildung 2 zu sehen, umfasst die Dokumentation in maschinenlesbarer Form den Zeitpunkt, den Benutzer, die Upload- oder Löschoperation und die Image-, Study- und Subject-UID. Dadurch wurde das DICOM-Upload-Modul erfolgreich um eine Funktion erweitert, die es ermöglicht, relevante Daten beim Upload gemäß den gesetzlichen Bestimmungen zu dokumentieren.

3.5 GitLab URLs

Aziz, Emrullah, Tom

Ausgangssituation Die Erstellung von Deep Learning Trainings auf dem Cluster benötigt eine Reihe von Dateien zur Konfiguration des verwendeten Netzes sowie Preprocessing der verwendeten Daten. Diese können durch das Cluster-Dashboard in einer simplen Webanwendung angegeben werden. Um Dateien zum Start eines Trainings verwenden zu können, müssen diese auf Netzlaufwerken hochgeladen werden, damit der gestartete Prozess auf dem Cluster auf diese zugreifen kann. Dies bedeutet meist eine unübersichtliche Anzahl an Dateien in Online-Speichern und Probleme bei der Verwaltung und Versionierung verwendeter Konfigurationen.

Aufbau des Projekts Bei der Inklusion von GitLab Links geht es primär um die Planung und Implementierung der Funktionalität, Trainings über das Cluster-Dashboard mit auf GitLab gespeicherten Dateien zu starten. Dies soll per Link, und, wenn nötig, zusätzlich per GitLab Access Token (falls ein privates Repository verwendet wird), geschehen. Das heißt, es müssen existierende Eingabefelder so überarbeitet, beziehungsweise neue erstellt, werden, dass das Cluster-Dashboard für die `config.py` (Konfigurationsdatei für das verwendete Netz) und für die Modell- und Architektur-Dateien die Webadresse einer Datei aus einem GitLab Repository einlesen kann. Nach Übergabe an das Girder-Backend kann dieses mit einem dazugehörigen Access Token die gewünschten Dateien auf den Cluster-Node herunterladen, um mit diesen ein Training zu starten.

Um diese Funktionalitäten zu implementieren gibt es primär vier Programme und Services, an mit denen gearbeitet werden muss. Das erste wäre das intern laufende

Hashicorp Vault. Das zweite Element ist eine Implementierung von GitLabs API, die es ermöglicht, die Dateien aus einem Repository herunterzuladen, damit diese für Trainings genutzt werden können. Die letzten beiden Elemente sind das Front- und Backend in Form des Cluster-Dashboard und der Girder-Instanz, welche überarbeitet werden müssen, um Eingaben für GitLab-URLs und die Nutzung der vorherig genannten API zu ermöglichen.
Anfänglich musste dafür gesorgt werden, dass das Cluster-Dashboard und Girder zusammen lokal gestartet werden können. Dies dient dem Testen der Elemente die wir implementieren, ansonsten könnten wir nicht sehen welche Auswirkung unsere Änderungen bzw. Erweiterungen am Code haben. Um dies zu ermöglichen haben wir die nötigen Projekte mit zugehöriger Anleitung zum lokalen Starten zur Verfügung gestellt bekommen, weswegen wir lediglich weitere Dateien definiert von der Anleitung anlegen mussten und einige Umgebungsvariablen setzen mussten. Dazu musste auch noch das FME MEVIS Zertifikat zur Authentifizierung mit anderen Hintergrundservices eingebunden werden, der Vorgang folgte dabei größtenteils verschiedenen vorgegebenen Anleitungen.

Beim ersten Element, Hashicorp Vault, handelt es sich um ein sicheres Speichersystem für sensible Informationen wie API-Schlüssel, Passwörter, Zertifikate und ähnlichem. Somit dient Vault zur Sicherung solcher Daten und Informationen. Im Cluster-Dashboard wird Vault ebenfalls benutzt. Hier werden Vault Tokens bereitgestellt, die dem Nutzer Zugriff auf seine abgelegten Informationen geben. Wir haben nun vor, zusätzlich auf Vault die GitLab Tokens, die zum Abrufen der gewünschten Dateien genutzt werden, dort zu speichern für einen einfacheren und sicheren Zugriff.
Das zweite Element, die API, muss so implementiert werden, dass die Dateien von dem angegebenen Repository nicht nur heruntergeladen werden können um sie für das Starten eines Trainings nutzen zu können, sondern auch dass sie die Dateistruktur abrufen kann, damit wir diese in der Nutzeroberfläche des Cluster-Dashboards anzeigen können und Dateien über eine Baumstruktur ausgesucht werden können (dazu mehr später).
Für die API zum Zugriff auf in GitLab gespeicherten Daten haben wir ein existierendes Skript bekommen, welches die Dateien herunterladen kann. Dieses

kann in Girder während der Vorbereitung eines neuen Trainings als eigene im voraus gestartete (`prestart`) Nomad-Task hinzugefügt werden und so die benötigten Dateien auf die Instanz auf dem Cluster herunterladen.
Um eine Baumansicht der existierenden Dateien auf die ein Nutzer Zugriff hat zu ermöglichen, haben wir in Girders REST-API einen neuen Endpunkt defininert, über welchen die verschiedenen Inhalte eines GitLab Projektes abgefragt werden können, ohne erhöhte Kosten durch nicht benötigte Downloads zu erzeugen. Dies liegt daran, dass wenn wir über die existierende API Baumstrukturen ausgeben wollen würden, die das zugehörige Repository repräsentieren, so müssten wir alle nötigen Dateien pullen, also herunterladen, und diese in der Baumstruktur einordnen. Dies ist zum einen speicherintensiv, da manche Repositories möglicherweise mehrere hunderte von Dateien haben kann und diese alle erst heruntergeladen werden müssten und zum anderen aufwendig, da man von diesen hunderten Dateien noch die Namen alle einzeln entnehmen müsste und in die Baumstruktur einordnen müsste.
Als letztes Element ist das Cluster-Dashboard als Frontend für die nutzerfreundliche Erstellung von Trainings neuronaler Netze zuständig. Dies ist eine von MEVIS selbstentwickelte Webanwendung, die größtenteils in Vue.js mit Quasar entwickelt wurde. Eingegebene Daten werden an Girder, einer im Hintergrund laufenden Daten-Management Plattform, übermittelt, wo sie von eigens geschriebenen Erweiterungen interpretiert und zur finalen Ausführung an das Nomad-Cluster geschickt werden.
Das Cluster-Dashboard sah zu Beginn so aus:

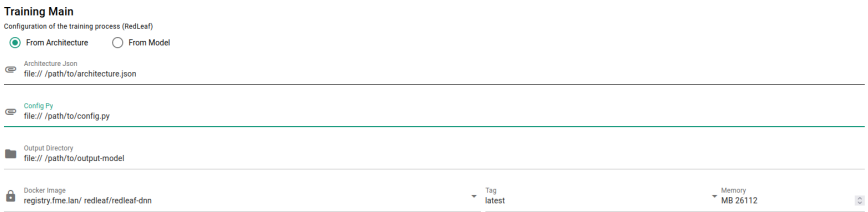


Abbildung 3: Screenshot des Klammer-Interface

Im ersten Bild sieht man das Trainings-Interface, in welchem man die Dateien für das Training per manueller Eingabe auswählen kann und weitere zusätzliche Einstellungen bezüglich des Trainings. Im zweiten Foto sieht man das Klammer-Interface, bei welchem man zwischen Personal oder Global Collection Dateien

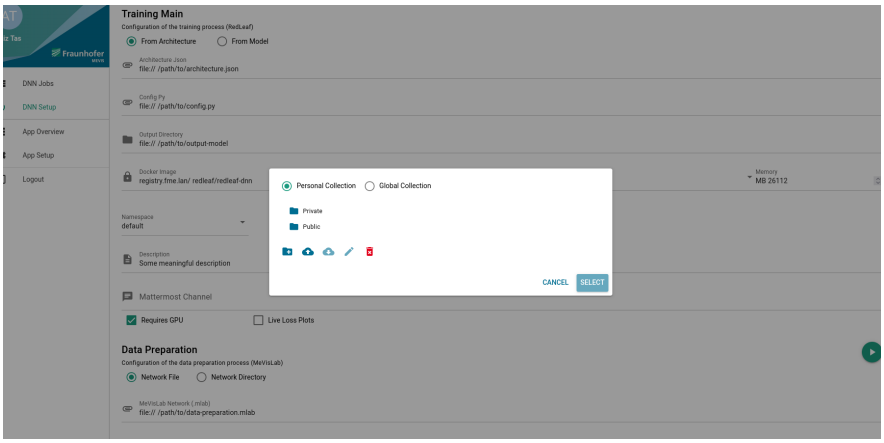


Abbildung 4: Screenshot des Trainings-Interface

aussuchen kann.

Zuerst haben wir geplant einen weiteren Radio-Button wie from `Architecture.json` anzulegen bei dem man alle nötigen Eingaben für das herunterladen von Dateien von GitLab-Repositories findet. Da jedoch bereits ein Dialogfenster per Klick auf das Klammersymbol neben dem Eingabefeld implementiert war, haben wir uns dazu entschieden innerhalb dieses Klammer-Interfaces eine weitere Auswahlmöglichkeit für die Eingabe von GitLab Links und Token einzufügen. Zusätzlich wollten wir eine Baumstruktur einfügen, die die Dateistruktur des zum eingegebenen Link zugehörigen Repository darstellt, ähnlich wie bei der global und personal Collection(siehe Bild 2).

Um diese Elemente zu implementieren müssen wir die vorhandenen Vue Dateien bearbeiten bzw. falls nötig neue Vue Dateien anlegen, die die nötigen Inputs erzeugen. Diese Inputs müssen dann nachdem die nötigen APIs in den Dateien eingebunden si.envnd mit diesen verbunden werden, sodass die eingegebenen Inputs von den APIs verarbeitet werden und die im Input angegebenen Dateien heruntergeladen werden.

Nun sieht es so aus:

Hier wurde eine neue Auswahlmöglichkeit für GitLab-Links eingefügt. In diesem Fenster befinden sich zwei neue Inputs: Ein Input für den GitLab Repository Link und ein Input für den dazugehörigen Token. Zusätzlich gibt es hier auch noch einen Knopf mit der Aufschrift "Fetch Data", der nur gedrückt werden kann, wenn der Input für den Link nicht leer ist. Wird nur ein Link eingegeben, versucht dem hier genutzten HTTP-Client Axios die angegebene Datei zu pullen

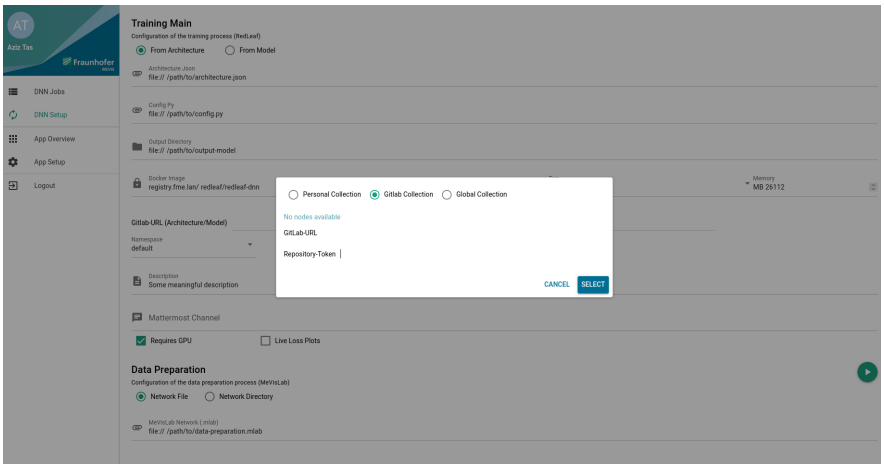


Abbildung 5: Screenshot der neuen Inputmöglichkeit

und gibt eine Fehlermeldung aus, wenn ein Token benötigt wird bzw. wenn ein Fehler beim Herunterladen der Datei auftritt. Dies gilt auch für den Fall, wenn beide Eingabefelder gefüllt sind, der Token oder Link jedoch ungültig ist. Dazu sollte auch eine Datei-Baumstruktur ausgegeben werden, falls der eingegebene Link nicht auf eine Datei zeigt sondern auf ein ganzes Repository. Die Eingaben werden über eine zusätzliche Vue Datei erzeugt in der sich auch alle Funktionen bezüglich der APIs befinden, um Fehler und Unübersichtlichkeit zu vermeiden.

Während der Implementierung gab es mehrere Probleme auf die wir gestoßen sind, wobei es grob zwei Arten von Problemen gab, die oft auftraten. Die erste Art sind fehlende Kenntnisse und Erfahrung, da viele von den zu programmierenden Elementen entweder über ein Medium programmiert werden mussten, über welches wir nicht viel wussten oder nicht wussten wie es programmiert werden kann. Die andere Art von Problem auf die wir oft gestoßen sind, sind Kompilierungsfehler. Diese waren meist leicht zu beheben, jedoch hatten wir auch welche an denen wir über einen längeren Zeitraum arbeiten mussten um den Fehler zu beheben. Diese Fehler mussten auch nicht sonderlich komplex gewesen sein, ein paar von den waren nur mit einer Zeile Code zu lösen, jedoch ist es vor allem bei größeren Projekten mit vielen Zeilen Code in mehreren Dateien mitunter schwer, einen solchen Fehler zu finden und zu beheben.

3.6 Netzkonfiguration via MeVisLab

Arbresh, Jörn, Felix

Das Ziel dieses Teilprojektes bestand darin Nutzer*innen eine graphische Oberfläche zu bieten. Diese soll es ermöglichen die Konfigurationsdateien, die für das Training eines neuronalen Netzen benötigt werden, im MeVisLab per Dateiauswahl auszuwählen.

Bisher war es notwendig per Kommandozeile beim Starten eines Trainings, sowohl lokal als auch auf dem Cluster, die Konfigurationsdateien je per Flag hinter der eigentlichen Eingabe mit Dateipfad anzugeben.

```
rld-training_main -a ./architecture.json -c ./config.py out_debug
```

Für ein Training sind zwei verschiedene Konfigurationsdateien nötig, eine Json Datei, welche die Architektur spezifiert und eine Python Datei, die die Hyperparameter des zu trainierenden Netzes festlegt. Dabei musste man oft Dateien über diverse Unterordner hinweg heraussuchen.

Das Layout von dem TrainingConfig Modul ist simpel und übersichtlich gestaltet. Im Grunde können Nutzer*innen zwischen zwei Tabs hin und her wechseln, wobei die Tabs das Konfigurieren für jeweils einen der beiden Dateien steht. Außerdem gibt es innerhalb der Tabs die Möglichkeit die Quelle für die Konfiguration auszuwählen, der nächste Abschnitt gibt einen kurzen Einblick darüber.

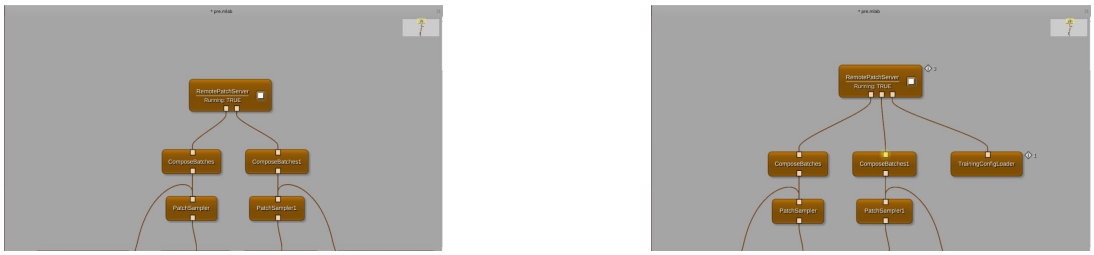
Das neue Modul bietet neben der Auswahl bereits existierender Konfigurationsdateien an, im Modul die Konfigurationen (jeweils für die Architektur und für die Hyperparameter) dort selbst anzulegen. Hierfür gibt es für beide Konfigurationen einen Editor, wo mittels Python oder Json konfiguriert werden kann, je nach dem in welchen Tab man sich befindet. Der Editor übernimmt sofort die Änderungen die vorgenommen wurden und aktualisiert seine Ausgabe, wodurch nichts ein weiteres Mal bestätigt oder angeklickt werden muss. Der Editor kann ziemlich hilfreich sein, um verschiedene Konfigurationen auszuprobieren, da man hier nur das Modul öffnen muss und seine Änderungen hineinschreibt.

Die Quelle der Konfiguration lässt sich intuitiv ändern, indem man innerhalb des Tabs eine Combobox anklicken kann und die gewünschte Quelle auswählt. Dadurch verschwindet der Editor und es erscheint ein Filebrowsing Feld, womit man seine Konfigurationsdatei auswählen kann. Das Ändern der Quelle für die Konfiguration bewirkt automatisch die Änderung des Outputs.

Für die Verwendung des Moduls TrainingConfig wird nach der Auswahl der Konfigurationsdateien das Modul mit dem RemotePatchServer verbunden, welcher einen Server für den Trainingsprozess bereitstellt. Die Konfigurationsdateien werden als Streams übertragen und können im weiteren Verlauf für das Training genutzt werden.

In diesem Abschnitt soll ein kleiner Überblick darüber gegeben werden, welche Komponenten in der Trainingskette/struktur verändert worden sind, um das Modul zu realisieren. Wie vorher bereits erklärt, stellt das Modul RemotePatch-Server einen Server bereit, welcher von dem Trainingsprozess benutzt wird um die Trainingsdaten abzugreifen. Dadurch, dass wir die Konfiguration nun über ein Modul einstellen, ist der RemotePatchServer die einzige Schnittstelle zu dem Trainingsprozess. Daher ist es naheliegend, dass das TrainingConfig Modul seine Informationen an den RemotePatchServer übergeben muss, damit dieser diese ebenfalls bereitstellt.

Um TrainingConfig an RemotePatchServer binden zu können, musste zuerst ein weiterer Anschluss im Skript des Moduls (RemotePatchServer) hinzugefügt werden. Damit die eingehende Konfiguration behandelt und dem Training bereitgestellt werden kann, mussten dementsprechend Funktionen im Skript implementiert werden. Hier hört es nicht auf, denn das Training muss umgeschrieben werden, sodass es erst einmal die Konfiguration von dem Server einholt und zusätzlich muss es mit dem Code der Konfigurationen als Input umgehen können (da der Code im Editor logischerweise keinen Pfad hat, sondern direkt übergeben wird). Diese Anforderungen haben dazu geführt, dass einige Funktionen im Training dementsprechend angepasst oder gar neue Funktionen geschrieben werden mussten.



(a) vorher (b) nachher

Abbildung 6: Netzkonfiguration in MeVisLab

Um bestehende Arbeitsabläufe nicht einzuschränken, wurde das Modul (TrainingConfig) so integriert, dass die in den Abbildungen gezeigte Verwendung der Flags -a und -c, weiterhin gleichermaßen funktioniert. Bei der simultanen Nutzung des Moduls und der Flags werden die angegebenen Konfigurationsdateien hinter den Flags verwendet.

Die Integration der Konfiguration von Netzarchitekturen und Hyperparametern in die grafische Benutzeroberfläche von MeVisLab stellt einen bedeutenden Fortschritt dar. Dieses neue Feature ist bereits implementiert und nutzbar, was den Workflow erheblich verbessert. Durch die GUI wird die Steuerung und das Starten von Trainingsprozessen vereinheitlicht und vereinfacht, da die entsprechenden Befehle generischer und benutzerfreundlicher gestaltet sind.

Auf Basis dieser Arbeit ist es zukünftig möglich, weitere Module zu entwickeln, die spezifische Architekturvorlagen bereitstellen. Diese Module können die Konfiguration vollständig durch interaktive Dialoge ersetzen, was die Notwendigkeit, Textdateien manuell zu bearbeiten, überflüssig macht. Dies stellt eine signifikante Verbesserung für Benutzer*innen dar, die diese sich nun stärker auf die Optimierung und Anwendung ihrer Netzwerke konzentrieren können, anstatt sich mit den technischen Details der Konfiguration auseinandersetzen zu müssen.

Zusammenfassend lässt sich sagen, dass diese Erweiterung durch das *TrainingConfig* Modul nicht nur die Benutzerfreundlichkeit erhöht, sondern auch die Effizienz und Flexibilität im Umgang mit verschiedenen Netzarchitekturen und Hyperparametern steigert. Dies legt den Grundbaustein für zukünftige Entwicklungen und Anpassungen, die die Arbeit mit neuronalen Netzwerken weiter vereinfachen und optimieren werden.

4 Deep Learning

Jessica

Das Thema rund um künstliche Intelligenz und Neuronale Netze hat in den letzten Jahren sehr an Reichweite gewonnen. So behaupten LeCun et al., 2015, dass Deep Learning in der näheren Zukunft auch noch viele weitere und neue Erfolge erzielen wird. Viele neu aufkommende Probleme als auch bereits bestehende Probleme werden versucht mithilfe tiefer Neuronaler Netze zu lösen.

Auch im medizinischen Bereich ist der Einsatz von Deep Learning Methoden bereits etabliert. Unter anderem werden hier Modelle trainiert, welche die Segmentierung von Organen oder anderen Körperteilen auf CT oder MRT-Scans übernehmen. (Shen et al., 2017)

In Kooperation mit dem Universitätsklinikum Hamburg-Eppendorf (UKE) wurde ein Projekt zur Quantifizierung von Sarkopenie initiiert, dessen Ziel es ist, die diagnostische Genauigkeit zu verbessern und die Forschung zur Muskelsegmentierung durch den Einsatz Neuronaler Netze voranzutreiben. Unser Fokus lag auf der Nutzung von Neuronalen Netzwerken zur Erkennung und Analyse zweier relevanter Muskelstrukturen, der sogenannten Bauchwandmuskulatur und der paravertebralen Muskulatur.

Im Weiteren wird sowohl die Krankheit Sarkopenie als auch das Pre- und Postprocessing des Trainings erläutert. Hierbei wurde auf Trainingsdaten mit SPLAT Annotationen trainiert. Diese Art von Annotationen wurde in Abschnitt 3.2 genauer erläutert und deren Vorteile verdeutlicht.

4.1 Sarkopenie

Hannah

Sarkopenie ist eine Krankheit, die durch einen altersunabhängigen generellen progressiven Verlust von Muskelmasse und -kraft gekennzeichnet ist.

Besonders prävalent ist diese in der älteren Bevölkerung. Durch den Verlust von Muskelkraft kann es für Betroffene zu starken Einschränkungen und folgend einem Verlust von Lebensqualität kommen, auch steigen mit dem Fortschritt der Krankheit die Hospitalisierungs- und Mortalitätsraten.

Ohne angemessene Therapie und Behandlungsmaßnahmen führen diese Symptome und Folgen oft zu einem rapideren Fortschritt der Krankheit, ein sich selbst verstärkender Kreislauf. Durch eine frühe Diagnose und entsprechende Therapien kann einem Fortschreiten der Krankheit effektiv entgegengewirkt werden. Durch gezielte Therapie kann die vorhandene Muskelmasse und -kraft erhalten werden oder sogar wieder aufgebaut werden und der Patient kann verlorene Kraft und Lebensqualität erhalten oder wieder erlangen (Hurezeanu2, 2023).

Um so eine Behandlung zu ermöglichen ist eine Diagnose notwendig, hier werden im klinischen Alltag oft Maße der Muskelkraft wie die Griffstärke, oder die Zeit die ein Patient benötigt, um fünf Mal aus dem Sitzen ohne Nutzung der Arme aufzustehen verwendet. Beides sind gute Prädiktoren für generelle Muskelkraft und Gesundheit, die keine teure Technik benötigen (Cruz-Jentoft et al., 2019).

Eine alternative Art der Diagnose, die präziser ist und auch genutzt werden kann wenn andere Tests nicht möglich sind z.B. auf Grund von einer Paralyse, ist die Erfassung der Muskelmasse auf bildgebenden Verfahren wie der Computer Tomographie (CT). Besonders bietet sich die Region des dritten Lendenwirbels (L3) an, da die Muskelmasse in diesem Bereich eine besonders starke Relation zur generellen Muskelmasse und Gesundheit vorliegt (Mourtzakis et al., 2008).

Wir haben uns deshalb zur Aufgabe gemacht ein neuronales Netz zu trainieren, das die Muskeln im Bereich zwischen dem zweiten und vierten Lendenwirbel (L2-4) erkennt und die Masse dieser berechnen kann. In den nächsten beiden Kapiteln erläutern wir, wie wir die Daten für das Training vorbereitet und anschließend die Ergebnisse des Neuronalen Netzes nachbearbeitet haben.

4.1.1 Pre-Processing

Jessica, Hannah

Datenvorbereitung - SPLAT Annotation Um ein Neuronales Netz zu trainieren ist es notwendig, die vorhandenen Daten entsprechend vorzubereiten. Wie in Abschnitt 3.2 bereits beschrieben, werden die Daten in drei disjunkte Gruppen aufgeteilt - Trainings-, Validierungs- und Testdaten. Für das Pre-Processing wird das Framework MeVisLab verwendet. Die `.mlab`-Datei, welche die Pre-Processing-Schritte enthält, wird bei dem Trainingssetup mittels Pfadangabe eingebunden. Die annotierten Daten laufen vor dem Trainingsbeginn die definierten Schritte durch und werden so bearbeitet, dass das Netz aus den eingegebenen Daten möglichst viel lernt.

Hierbei werden nur die Gruppen Training und Validierung betrachtet. Die Gruppe Test wird genutzt, um das Netz nach dem Training auf komplett ungesehenen Daten zu evaluieren und somit bei der Evaluierung ein möglichst aussagekräftiges Ergebnis zu erzielen.

Durch die mit SPLAT annotierten zwei Muskelstrukturen Daten benötigt das Neuronale Netz, beziehungsweise die Loss Funktion zum Berechnen des Losses weitere Informationen über die Gewichtung der annotierten Voxel. Da beide Strukturen einander ausschließen, konnte eine einfache Logik eingebaut werden. Hierzu betrachte man die Annotationen der beiden Strukturen pro Voxel:

- Wenn eine Struktur Vordergrund ist, ist im Voxel weder die andere Struktur noch Hintergrund vorhanden.
- Wenn beide Strukturen Hintergrund sind, ist im Voxel keine der beiden Strukturen vorhanden.
- Sollte der Voxel auf beiden Strukturen nicht annotiert sein, ist nicht eindeutig was im Voxel vorhanden ist. Dieser Voxel wird bei der Loss-Berechnung ignoriert.

In Anbetracht dieser beiden Regeln werden die Gewichte der Voxel gesetzt, sodass die Loss-Funktion die nicht annotierten Voxel ignoriert und die annotierten Voxel logisch richtig mit in die Berechnung des Losses einbezieht.

Da wir nur eine relativ kleine Menge Trainingsdaten haben, nutzen wir zwei Methoden um unsere Daten zu augmentieren. Zuerst Spiegeln wir alle Bilder, was nur möglich ist, da die uns interessierenden Strukturen im Körper symmetrisch sind. Als zweites werden alle Bilder zufällig leicht rotiert. So können wir unsere Daten vollständig ausnutzen und das Netz kann mit einer größeren Diversität von Daten lernen.

Dem Netz wurde in einem Durchlauf nicht eine komplette CT-Scan Schicht gegeben, sondern wir haben eine CT Schicht in sogenannte Patches geschnitten und diese an den Seiten ausgepolstert. So können wir die Menge der Trainingsdaten weiter erhöhen und sicherstellen, dass alle Ausschnitte, die im Training verwendet werden dieselbe Größe haben. Für ein Training müssen wir diese auf die Architektur unseres Neuronalen Netzes anpassen, wobei das Padding dafür sorgt, dass die wichtigen Informationen in der Mitte des Bildes sind und so besonders viel Einfluss auf das Ergebnis haben.

4.1.2 Training

Hannah, Jessica

Trainingskonfiguration Trainiert wurde auf einem U-Net, welches für Segmentierungsaufgaben gut geeignet ist (Du et al., 2020). Hierbei kommen sogenannte Convolutional Schichten zum Einsatz, welche vor allem dafür sorgen, dass das Netz räumliche Anordnungen verstehen kann. Das U-Net benutzt einen 3×3 Filter, das ist die Filtergröße, die für solche Aufgaben bewährter Standard ist (Camgözlü und Kutlu, 2020) und fünf Level hat. Durch die Auflösung und Größe unserer Bilder war die Architektur des Netzes auf etwa fünf Level nach oben beschränkt, diese wurden vollständig ausgenutzt, um die Komplexität des Modells zu erhalten (Jena et al., 2023).
Ein weiterer Faktor war die Größe des Receptive Fields, dieses wollten wir relativ groß halten, damit unser Netz die größeren Zusammenhänge hinter den Strukturen besser lernen kann (Behboodi et al., 2020).

Wir haben auch drei Ausgabekanäle definiert - Bauchwandmuskulatur, paravertebrale Muskulatur und Hintergrund.

Wichtige weitere konfigurierbare Parameter für das Training mit einem Neuronalen Netz stellen die zu wählende Loss-Funktion wie auch die Lernrate dar. Als Lernrate haben wir einen Wert von 0.0001 gewählt, dieser bestimmt um wie viel Wert sich ein Gewicht pro Anpassungsschritt verändern darf. Bei der Loss-Funktion haben wir uns für eine Kombination aus zwei verschiedenen Arten von Funktionen entschieden. Zum einen den Dice Loss und zum anderen die Categorical-Cross-Entropy (CCE). Durch das Kombinieren dieser beiden Funktionen kann man Nachteile der Dice Funktion etwas ausgleichen. Dice Loss reagiert sehr empfindlich auf kleine diffuse Fehler, während CCE nicht zwischen großen und kleinen Fehlern unterscheidet und somit die gesamte Loss-Funktion etwas ausbalanciert (Zhang et al., 2021). Unsere Abbruchbedingung war eine Iterationsanzahl von 10000.

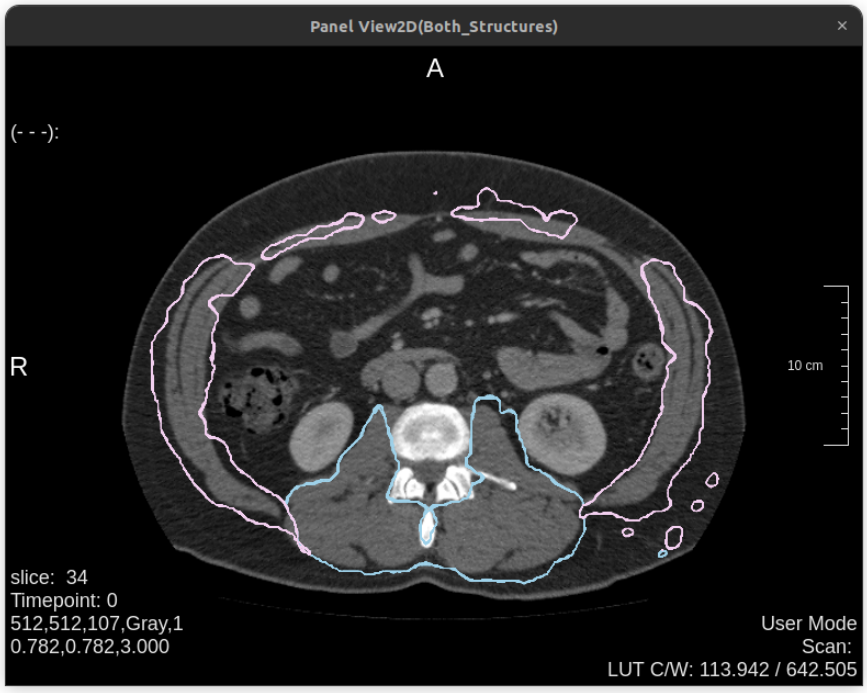


Abbildung 7: Anwendung des neuronalen Netz auf einen CT-Scan

Ergebnis Unser Netz erkennt grob die Bauch- und Rückenmuskulatur. Besonders gut wird die Rückenmuskulatur erkannt, sie werden etwas großzügig, aber meist komplett erfasst, mit wenig Fehlern. Auch die Bauchmuskulatur wird erkannt, diese aber oft nur teilweise und so großzügig, dass Teile von Organen

innerhalb der Bauchhöhle ebenfalls bei den Bauchmuskeln mit einbegriffen werden. Ebenfalls ist unser Netz bei den Übergängen zwischen Bauch- und Rückenmuskulatur noch unsicher und zählt Voxel in diesem Bereich öfter zu den falschen Muskeln.

Fazit Das Ziel, dass unser Netz Bauch- und Rückenmuskulatur erkennen kann, wurde erreicht, hier besteht aber noch einiges an Verbesserungspotenzial. Teilweise können die Fehler die unser Netz macht mit einem Post-Processing ausgeglichen werden, aber für ein optimales Ergebnis ist unser Netz zu schlecht. Hier bedarf es noch an einiger Experimentation und Optimierung unseres Modells.

Es wäre interessant zu sehen, wie die Tiefe und Filtergröße unseres Netzes das Ergebnis verändern, und wie weit verbesserte und mehr Annotationen der Trainingsdaten die Genauigkeit unseres Modells beeinflussen.
Auch wäre die Nutzung von TrainingLoop für ein aktiveres Lernen und Verbessern interessant, dafür hatten wir aber leider nicht genug Zeit.

4.1.3 Post-Processing

Hannah

Motivation Bei der Betrachtung der Ergebnisse des von uns trainierten Neuronalen Netzes (siehe 7) fällt auf, dass einige Fehler und Ungenauigkeiten bei vielen Fällen auftreten, besonders für solche Fehler ist eine Nachbearbeitung, die auch Post-Processing genannt wird, besonders sinnvoll. So können wir unsere Ergebnisse systematisch verbessern, ohne manuelle Verbesserungen für jeden einzelnen Fall durchzuführen. Größere Fehler wie nur teilweise erkannte Abdominalmuskeln können nicht mit Hilfe von Post-Processing gelöst werden, aber wir können kleine Löcher und Punkte entfernen, Muskelumrisse genauer erkennen und Fetteinschlüsse innerhalb der Muskeln ausschließen.

Dafür laden wir die Testfälle aus unserem Datensatz und wenden das von uns trainierte Neuronale Netz auf diese an. Bei der Betrachtung dieser Ergebnisse notieren wir häufig auftretende Fehler und versuchen anschließend, diese zu verbessern.

Zu Verbessernde Punkte:

- Strukturen werden zu großzügig erkannt
- Struktur wird nur teilweise oder zu knapp erkannt
- Löcher in erkannten Strukturen
- Kleinste Punkte, die grundlos erkannt werden

Umsetzung Da diese Probleme bei den Paravertebral- und Abdominalmuskeln unterschiedlich stark sind, behandeln wir diese in der Nachbearbeitung separat, um ein möglichst optimales Ergebnis für beide zu erzielen.

Als ersten Schritt filtern wir unsere Ergebnisse, dass nur die Voxel beibehalten werden, die einen Wert zwischen -29 und 150 Houndsfield Units haben, das ist der Wertebereich in dem alle Muskeln bei CT-Scans liegen. Dadurch werden z.B. Teile der Bauchhöhle entfernt, die durch zu großzügig erkannte Strukturen in unserem Ergebnis einbegriffen waren. So können wir effizient einen Großteil der falsch erkannten Bereiche von allen weiteren Operationen ausschließen und somit daraus resultierende Folgefehler minimieren (siehe Abbildung 8).

Um kleine unbeabsichtigte Löcher zu füllen, nutzen wir eine morphologische closing Operation. Da die Löcher, die wir im Vorhinein beobachtet haben, meist relativ klein sind, haben wir uns dafür entschieden nur einen 3×3 Kernel zu benutzen (Siehe Abbildung 9).

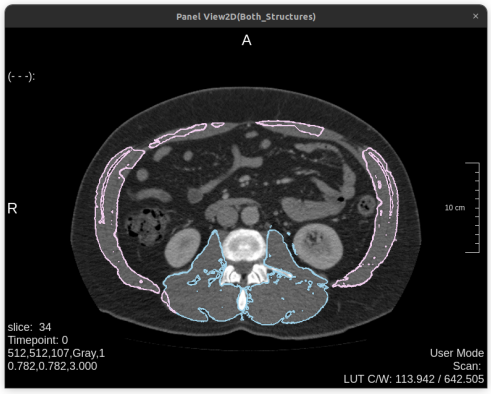


Abbildung 8: Ergebnis nach Anwendung des Houndsfield Unit Filters

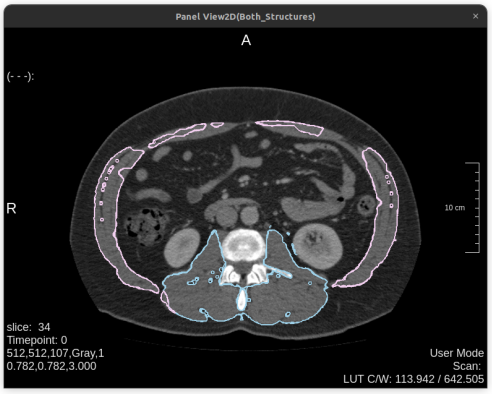


Abbildung 9: Ergebnis nach Anwendung des Closing Kernels

Kleine Punkte, die unzusammenhängend erkannt wurden, werden daraufhin mit dem `ConnectedComponentsMacro` entfernt. Dieses berechnet die Volumen aller zusammenhängenden erkannten Elemente und filtert alle Elemente raus, die kleiner als ein festgelegter Grenzwert sind.
Da unser Modell nur auf einer zweidimensionalen Basis Strukturen erkennt, haben wir uns hier ebenfalls dafür entschieden, die Volumen nur pro Schicht und nicht das Volumen über alle Schichten zu berechnen.
Auch haben wir uns für die Volumeneinheit Milliliter entschieden. Diese ist im Gegensatz zu einer Berechnung der Menge an Voxeln unabhängig von der Auflösung der Scans und ermöglicht es, sinnvolle Grenzwerte zu finden, die über alle Fälle gleich sind. Wir haben einige Grenzwerte ausprobiert und haben für die Rückenmuskeln einen Grenzwert von $0,5ml$ als optimal befunden. Die erkannten Strukturen sind meistens zusammenhängend und die relevanten Bereiche sind hier in der Regel größer als $0,5ml$. Die Bauchmuskeln sind im Gegensatz dazu oft in kleinere Segmente geteilt, wodurch bei einem Grenzwert von $0,5ml$ viele relevante Bereiche ausgeschlossen werden. Den Grenzwert haben wir hier nach einigen Abwägungen auf $0,1ml$ heruntergesetzt, es ist aber bei dem Stand von unserem Modell schwer, einen optimalen Grenzwert zu finden.
Dadurch, dass die erkannten Strukturen so unzusammenhängend und somit auch klein sind, können wir den Grenzwert nicht sehr hoch setzen. Dies führt dazu, dass größere Bereiche, die falsch erkannt wurden, schnell über dem Grenzwert liegen. Mit einem besseren Modell könnten vermutlich höhere Grenzwerte gewählt werden. Auch interessant wäre es zu sehen, ob bei einem Netz, was auf 3D Strukturen trainiert wurde, die dreidimensional berechneten Volumen ein besseres Ergebnis liefern.

Eines unserer großen Probleme ist, dass unser Modell oft Teile unserer Strukturen nicht erkennt. Dies ist besonders auffällig bei der Bauchmuskulatur, bei der oft die anterioren Teile nur teilweise oder gar nicht erkannt werden.
Dem wirken wir nun, mit einer morphologischen Dilatation der Strukturen, etwas entgegen. Eine Gefahr, die bei der Dilatation immer besteht, ist zu viel Dilatation zu nutzen. Hierdurch werden Teile von anderen Muskeln oder Strukturen unbeabsichtigt zu unseren Strukturen hinzugezählt. Der Dilatation skernel muss also passend gewählt werden.

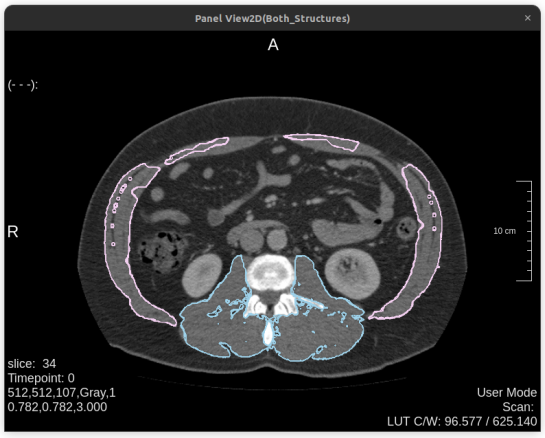


Abbildung 10: Ergebnis nach Anwendung des ConnectedComponentsMakro

Die Kernelgrößen, die wir gewählt haben, unterscheiden sich wieder für unsere zwei Strukturen. Da bei den Rückenmuskeln nur kleinere Fehler ausgeglichen werden sollen, haben wir für diese einen kleineren Kernel als für die Bauchmuskulatur, mit einer Größe von $3 \times 3 \times 3$, gewählt. Auffällig ist, dass wir hier einen 3D-Kernel benutzen und keinen 2D-Kernel, wie er für das Closing verwendet wurde. Da zu dem Zeitpunkt, an dem die Dilatation stattfindet, falsch erkannte Bereiche schon so gut wie möglich entfernt wurden, wollen wir hier die übrigen Bereiche möglichst weit ausweiten, ohne sich zu weit von den erkannten Strukturen zu entfernen. Weshalb wir hier die dritte Dimension hinzugezogen haben.
Für die Bauchmuskulatur haben wir uns für einen $5 \times 5 \times 3$ Kernel entschieden, um die umliegenden Bereiche möglichst großzügig einzubeziehen. Oft werden dabei auch Teile der paravertebralen Muskeln mit erkannt. Das korrigieren wir in einem späteren Schritt.

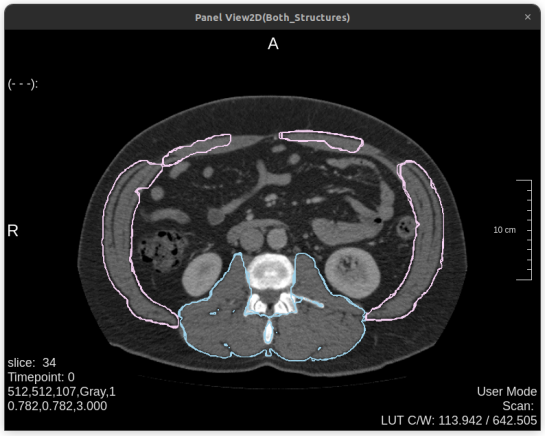


Abbildung 11: Ergebnis nach Dilatation

Der erste Schritt wird nun auf den überarbeiteten Masken wiederholt und alle Voxel, die keinen Houndsfield Unit Wert zwischen -29 und 150 haben, werden von den Strukturen entfernt. Da bei dem Closing und der Dilatation Bereiche, die keine Muskeln sind zu den Strukturen hinzugefügt wurden, die wir nicht in unserem Endergebnis haben wollen, konnten wir so diese Strukturen entfernen.

Ebenfalls führen wir das `ConnectedComponentsMacro` noch einmal auf beiden Strukturen aus. Für die paravertebralen Muskeln behalten wir auch den Grenzwert von $0,5ml$ bei. Für die abdominalen Muskeln können wir den Grenzwert auf $0,2ml$ hoch setzen, da durch Dilation die Strukturen jetzt besser zusammenhängen und auch größer sind.

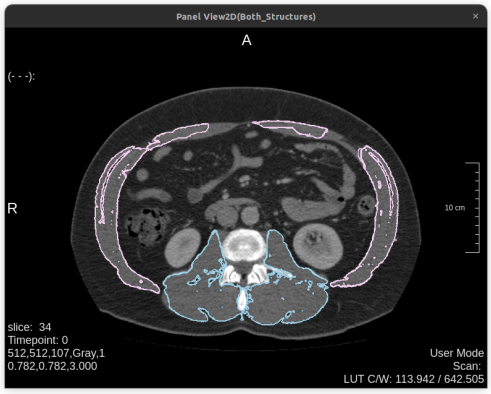


Abbildung 12: Ergebnis nach erneuter Filterung der Houndsfield Units

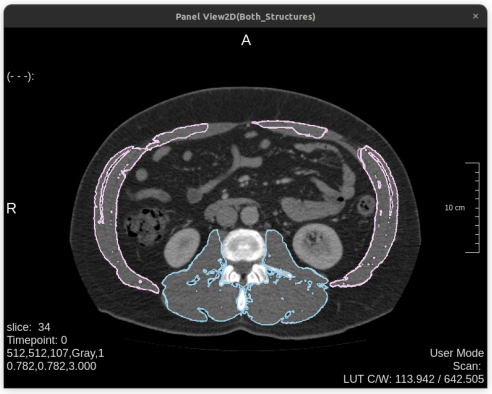


Abbildung 13: Ergebnis nach Anwendung des ConnectedComponentsMakro

Als finalen Schritt stellen wir sicher, dass jeder Voxel im Bild maximal zu einer Struktur gezählt wird.
Besonders durch die Dilatation kommt es vor, dass sich die beiden Strukturen überschneiden. Da unser Modell die paravertebralen Muskeln generell besser erkennt, entfernen wir aus der aktuellen abdominalen Muskel Maske alle Voxel, die ursprünglich von unserem Netz als paravertebral erkannt wurden. Die resultierende Maske wird schließlich von den bearbeiteten paravertebralen Muskeln abgezogen und die finalen Muskelklassifikationen werden wieder zusammengeführt. Wodurch wir unser Ergebnis sinnvoll speichern können.

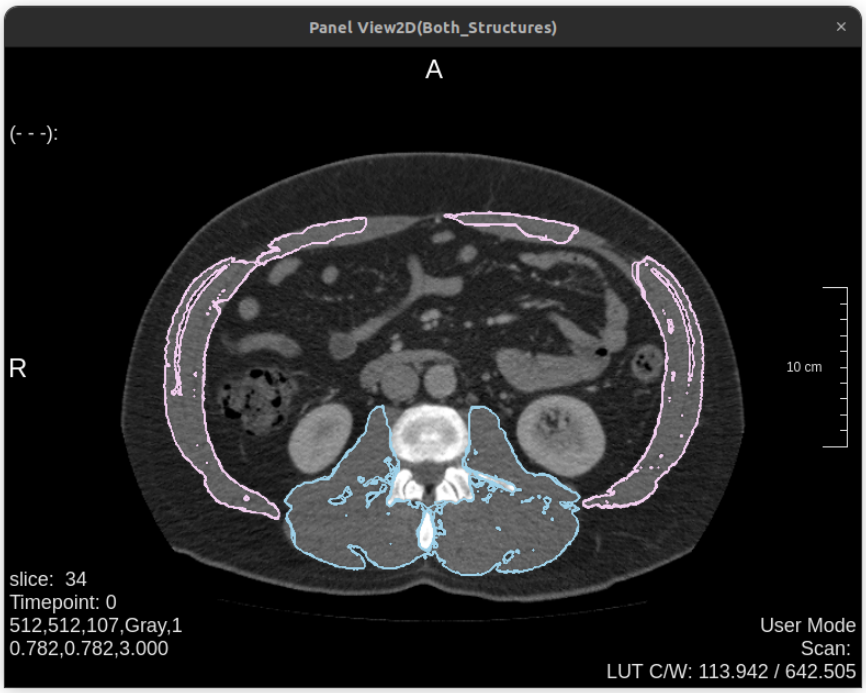


Abbildung 14: Finales Ergebnis nach dem Post-Processing

Fazit Das finale Ergebnis 14 ist sichtbar besser als unser Ausgangspunkt 7. Strukturen werden genau erkannt, einige falsch erkannte Punkte wurden entfernt, Löcher innerhalb der Strukturen beschränken sich auf Regionen von intramuskulärem Fett und Teile der Strukturen, die nicht komplett erkannt wurden, konnten teilweise zu unseren Strukturen hinzugefügt werden.

Auch auffällig ist, dass es dennoch weiterhin erhebliche Fehler in der Erkennung der Strukturen gibt. Besonders die Bauchmuskulatur wird meist nur teilweise erkannt, und Teile von Organen innerhalb der Bauchhöhle werden fälschlich erkannt. Ränder der Bauchmuskeln werden als Rückenmuskulatur interpretiert oder umgekehrt, oder die Rückenmuskeln werden nicht komplett erfasst. Dies sind Fehler, die die Möglichkeiten unseres Post-Processings übersteigen und nur durch ein besser trainiertes Modell behoben werden können.

Wir sind mit dem Resultat unseres Post-Processings insgesamt zufrieden. Das Ziel die Klassifizierungen unseres neuronalen Netzes zu verbessern haben wir eindeutig erreicht. Es gibt noch Potenzial es weiter zu verbessern und durch mehr Experimentation mit gewählten Werten oder anderen Modulen und Modulreihenfolgen unser Ergebnis weiter zu optimieren, aber wir haben hier einen natürlichen

Abschluss gefunden. Wir würden erst versuchen unser Modell weiter zu optimieren oder bessere Modelle trainieren, bevor wir die Nachbearbeitung verbessern.

4.2 Cluster-Dashboard

Jessica, Freja

Die Mitarbeiter*innen des Fraunhofer MEVIS verfügen über die Möglichkeit das Trainieren von Neuronalen Netzen (wie unter Deep Learning bzw. Pre-Processing beschrieben) auf dem Nomad Cluster durchzuführen. Das Cluster läuft auf den rechenstarken, internen Servern, welche die Möglichkeit bieten das Training eines Neuronalen Netzes durch geeignete GPUs wesentlich zu beschleunigen und lokale Ressourcen zu entlasten.

Um ein Training einfach und benutzerfreundlich auf dem Cluster zu starten, wurde das sogenannte Cluster-Dashboard entwickelt. Hier kann das gewünschte Training konfiguriert, gestartet und nun auch überwacht werden.

Die Benutzer*innen interessieren sich verständlicherweise für jegliche Informationen über das aktuelle Training. Diesbezüglich war ursprünglich vorgesehen gewesen, dass das Dashboard selbst in der Lage ist, den Nutzer*innen diese Informationen benutzerfreundlich zur Verfügung zu stellen. Es wurde allerdings schnell deutlich, dass die Tabs „Training Output“ und „Loss Plot“ keine nutzbaren Informationen boten, sondern stattdessen Fehlermeldungen anzeigten.

Unter dem Tab „Training Output“ sollten die Logs von Mevislab und dem Training zu finden sein. Der Tab „Loss Plot“ sollte eine Grafik des aktuellen Trainingsverlustes, sowie den berechneten Validierungsmetriken, mit einer Live-Update Funktion, anzeigen.

Die im folgenden vorgestellten Projekte befassen sich mit genau diesen beiden Punkten und stellen die gewünschten Funktionalitäten zur Verfügung. Es handelt sich dabei insbesondere um Arbeiten an der Benutzeroberfläche in dem bereits bestehenden `cluster/dashboard` GitLab-Repository vom Fraunhofer MEVIS.

4.2.1 Logger

Freja, Semjon

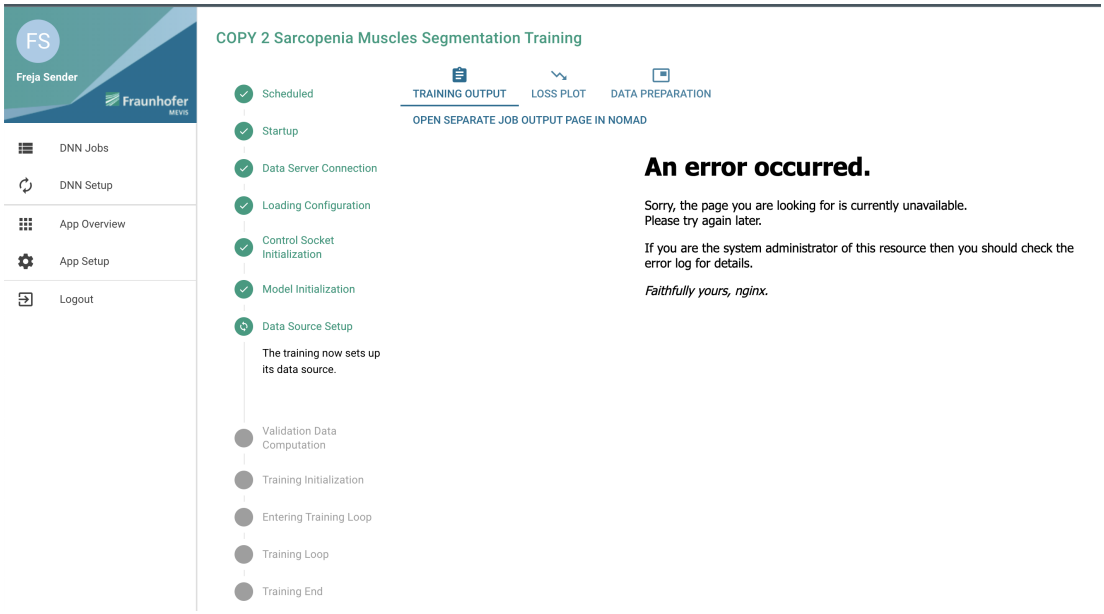


Abbildung 15: Screenshot des Logging-Bildschirms vorher

Motivation Die Motivation zur Überarbeitung des "Training Output"-Tabs ist in Abb. 15 deutlich zu erkennen: Es werden keine Informationen und aktuellen Logs des ausgewählten Trainings angezeigt, sondern ein Verweis auf die entsprechende Nomad-Webseite der Training-Logs, sowie eine Fehlermeldung.

Im Zuge dieses Teilprojektes war das Ziel, die Logs direkt in dem vorgesehenen Tab im Dashboard anzuzeigen. So sollten nicht nur die Logs des Trainings selbst, sondern auch jene der ausführenden Anwendung - MevisLab - verfügbar sein. Weiterhin sollte eine spezifische Allokation eines Jobs auswählbar sein und sowohl Logs der Standardausgabe wie auch die der Fehlerausgabe darstellbar sein.

Im Rahmen der vorangegangenen Funktionalität sollten die Logs außerdem in einem anderen System für einen definierten Zeitraum persistiert werden. Vorteil davon ist, dass auch die Logs von vergangenen, beendeten Allokationen noch für eine gewisse Zeit zur Verfügung stehen, selbst wenn diese von Nomad bereits aufgeräumt wurden. Dieser Zeitraum sollte laut Spezifikation in etwa eine Woche betragen.

Vorbereitungen Die Bestandsaufnahme der vorhandenen Infrastuktur im Fraunhofer MEVIS ergab, dass es sich um ein Nomad Cluster handelt, in welchem Docker Container laufen, die über verschiedenste APIs miteinander kommunizieren können.

Grundsätzlich wird in dem Kontext eines Trainings ein Job in Nomad angelegt, wie in der offiziellen Nomad Dokumentation (HashiCorp, 2024c) beschrieben. Dieser spiegelt die grundlegende Konfiguration des Trainings wieder. Die Allokation des Jobs wird durch Nomad auf einem beliebigen Knoten im Cluster erzeugt, sobald er gestartet wird. Diese wiederum besteht in unserem Fall aus einer sogenannten Task-Group, welche aus 3 Tasks besteht: Dem mevislab-sdk, training-main und dem vnc-streamer. Die Task Group wird von Nomad immer zusammen auf dem selben Knoten alloziiert. Ein weiterer Durchlauf des Jobs entspricht einer neuen Allokation. Dies ist in Abb. 16 mit zwei verschiedenen Allokationen zu sehen.

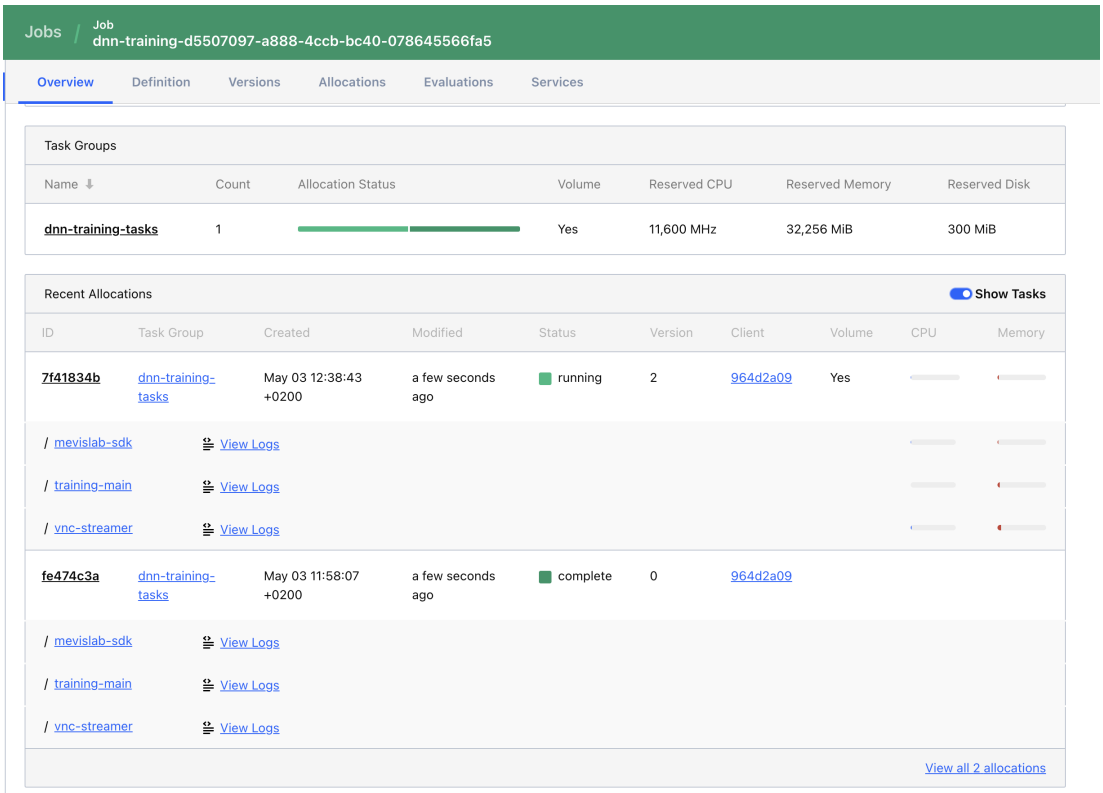


Abbildung 16: Übersicht eines Jobs in Nomad

Für die Aufgabenstellung wünschenswert wäre also ein System, welches automatisch alle im Cluster laufenden Trainings erkennt, sich dort die Logs abholt und in einer geeigneten Datenbank persistiert, um sie dann aus dem Dashboard-Client

wieder abrufen und anzeigen zu können. Nach anfänglicher Rücksprache mit Hans Meine wurde das sogenannte Log-Shipper-Pattern erwähnt, welches in unserer Recherche berücksichtigt wurde.

Die Recherche mündete schlussendlich darin, einen Stack mit Promtail (Grafana Labs, 2024) als Logshipper zu verwenden, welcher die Logs aus dem Docker Container einsammelt und in eine auf dem Cluster laufende Loki-Instanz pusht. Als erste Idee für eine UI könnte dann Grafana benutzt werden. Dieser Stack war unsere erste Wahl, da bereits positive Erfahrung in der Arbeit mit Grafana und großen Datenmengen bestand.

Lokales Proof-Of-Concept (Promtail-Loki-Grafana Stack) Im ersten Schritt wurde eine lokale Entwicklungsumgebung eingerichtet, welche der Produktivumgebung in den zentralen Punkten möglichst ähnlich sein sollte, ohne den Overhead aller Systeme zu haben. Dieser Punkt war von besonders großem Interesse um eine effiziente Arbeitsweise zu ermöglichen: Zunächst wurde ein lokales Nomad-Cluster aufgesetzt, was mit Hilfe der Anleitung (HashiCorp, 2024a) gelang. Hier wurde ein Flog-Promtail Job aufgesetzt, wobei Flog ein frei zur Verfügung gestelltes Programm ist, das Beispiel-Logs in konfigurierbaren Abständen produziert (Mingrammer, 2024).

Als zweite Task in der Taskgroup wurde Promtail aufgesetzt. Promtail war in diesem Fall der sogenannte "Log-Shipper", welcher die Logs aus dem /alloc/logs Ordner aus der Allocation lesen sollte. Unter Nomad teilen sich Tasks einer Task-Group innerhalb einer Allokation ein Verzeichnis. Der /alloc/logs Ordner ist ein solcher Ordner, in welchem Log-Files gespeichert werden (HashiCorp, 2024b).

Anschließend wurden Loki als Datenbank und Grafana als UI aufgesetzt. Problematisch war hierbei insbesondere die Kommunikation der einzelnen Komponenten untereinander, da in Nomad keine statischen IP Adressen zugewiesen werden, sondern diese sich - wie bei einem Container Orchestration System üblich - verändern, sobald der Task neu gestartet und neu alloziiert wird, worüber die Nutzer nicht die Kontrolle haben, sondern Nomad selbst.

Diverse weitere Probleme, rund um die spezifische Konfiguration der Software Komponenten wurden im weiteren Verlauf gelöst; diese Probleme waren größtenteils syntaktischer Natur und gingen mit der Fülle an Software-Funktionalitäten, die der Grafana Stack bietet, einher.

Zuletzt wurde der Fortschritt hinsichtlich dem ersten Stack allerdings verworfen. Als das lokale Proof-of-Concept soweit fertiggestellt war, fand Rücksprache mit dem DevOps-Team statt. Dort wurde leider deutlich, dass der Grafana-Stack im weiteren Umfeld der Organisation nicht flächendeckend einsetzbar war. Zugleich wurde von Seiten des DevOps Team an einer Lösung gearbeitet um sämtliche Logs im Unternehmen zentralisiert zu sammeln und zu persistieren. Dafür wurde ElasticSearch (bzw. OpenSearch) als Datenbank und Fluentd als Log-Shipper eingesetzt. Nach weiterer Rücksprache wurde von unserer Seite die Entscheidung getroffen, auf der Lösung vom DevOps Team aufzubauen – damit sowohl Kohäsion der Software-Landschaft innerhalb der Organisation, als auch Wartbarkeit neuer Software, die aus dem DeepAnatomy-Projekt entsteht, gefördert würde.

Cluster-Dashboard Entwicklung Die OpenSearch Infrastruktur wurde zu diesem Zeitpunkt in einer Beta-Version bereitgestellt. Nachdem uns Zugang gewährt wurde, begann die Einbindung in das Cluster-Dashboard.

Nach initialer Recherche zu einem bestehenden Node-Client für OpenSearch konnte keine funktionstüchtige, befriedigende Lösung gefunden werde, daher wurde dieser Client von uns direkt implementiert, sowie auch diverse Routinen zur Nachverarbeitung von empfangenen Daten.

Eine primäre Motivation des Cluster Dashboard - Loggers war es, Benutzbarkeit und Interaktivität zu gewährleisten, daher war sowohl Design als auch Styling von oberster Priorität. Ziel war eine Ansicht, welche große Mengen von textuellen Logs übersichtlich darstellt und zugleich auch stilistisch am Logger von Nomad angelehnt ist und daher vertraut aussieht.

Nach anfänglichen Besprechungen wurden die Grundfunktionen der gewünschten Software ausgearbeitet. Diese sollte sowohl Trainings- als auch MevisLab-Logs

separat darstellen können, verschiedene Arten von Logs (Fehler- oder Standardausgabe) getrennt oder gemischt ausgeben und zwischen verschiedenen Trainings-Sitzungen (Allocations) unterscheiden können. Auch eine visuell ansprechende Suche nach Schlüsselwörtern wurde als förderlich erachtet. Diese Funktionalitäten wurden allesamt erfolgreich implementiert. Im Sinne der Benutzbarkeit wurde ebenfalls ein "Load More"Button, welcher das reibungslose nachladen weiterer Logs ermöglicht, implementiert. Zuletzt wurde es als außerordentlich nützlich eingeschätzt sämtliche Such- und Filtereinstellungen in den URL-Parametern zu hinterlegen, sodass eine Ansicht per Knopfdruck kopiert und mit anderen Nutzer*innen geteilt werden kann. Wenn das Training gerade läuft werden die Logs in Realtime durch permanentes Polling geladen.

Ein signifikantes Problem stellte die Beschaffenheit der von OpenSearch bereitgestellten Daten dar. Grundsätzlich existiert ein Attribut, welches Aufschluss über den Auslieferungszeitpunkt gibt. Sollten diese Zeitstempel unter verschiedenen Logs allerdings identisch sein, so wie bei uns der Fall, liefert OpenSearch die Suchtreffer prinzipiell in einer zufälligen Reihenfolge aus.

Aus Zwecken der Optimierung werden Logs nicht einzeln, sondern gruppiert an OpenSearch ausgeliefert. Das ist im Sinne der Server-Belastung nachvollziehbar und notwendig. Client-seitig bietet sich in Folge dessen allerdings das Problem, diese Logs in der richtigen Reihenfolge darzustellen. So führte jeder Log ein Attribut mit sich, das Aufschluss über dessen Erstellung in der Datenbank gab, allerdings gab es Gruppen von etwa 50 Logs, welche intern den selben Erstellungszeitpunkt angaben. Das hatte zur Folge, dass ein erneutes Abfragen der neusten 100 Logs beispielsweise in einer zufälligen Reihenfolge mündete.

Diese Problem wurde schlussendlich implizit gelöst, allerdings ist es uns nicht möglich zu garantieren, dass alle Randfälle erfolgreich berücksichtigt wurden. Grundsätzlich wurde deutlich, dass die Seiten-basierte Abfrage der Logs, sortiert nach einem absteigenden Erstellungsdatum, das Problem behoben hat. Wurden also stetig 50 weitere Logs abgefragt und an die bereits dargestellten Logs angehangen, so wurde die ursprüngliche Reihenfolge, in der die Logs in OpenSearch eingespeist wurden, verwendet. Dabei war es für uns notwendig, die Reihenfolge dieser Logs innerhalb der Gruppierungen umzukehren, da diese in der Darstellung

im Dashboard in umgekehrter Reihenfolge angezeigt werden; der Grund dafür war stil-basiert, und hing mit dem verwendeten CSS-Styling zusammen. Unter der Zusammenarbeit mit dem DevOps-Team gab es diverse Anläufe durch Transformationen der Daten im LogShipper eindeutige Identifikationsnummern für jeden Log zu erzeugen, diese scheiterten allerdings.

Zum Schluss des Teilprojekts wurden retrospektiv bewährte Praktiken aus der modernen Frontend-Entwicklung umgesetzt, so zum Beispiel das konsistente Sammeln statischer Zeichenketten in Objekten, BEM-basiertes benennen von CSS-Klassen (getbem, 2024) und extrahieren der Markdown Inline Logik in „Computed"-Blöcke, um eine saubere Trennung zwischen Markdown und reinem Javascript Code zu schaffen.

An diesem Punkt fehlte die Funktionalität der Authentisierung gegenüber dem OpenSearch Endpunkt noch. Hier wurde der Entschluss gefasst, dass das bestehende Rechte- und Rollensystem effektiv genutzt werden könnte, sofern die Login Daten der entsprechenden Nutzer*in verwendet würden, welche sich die Logs über das Dashboard anschaut. Eine Option war es innerhalb der Anwendung die Base64-Verschlüsselten Authentisierungsdaten durchzureichen und mit jeder Anfrage an OpenSearch zu verschicken. Diese Authentisierungsdaten müssten dann in der „Local Storage" des Browsers hinterlegt werden. Dieser Ansatz wurde allerdings aus Sicherheitsgründen verworfen. Die Entscheidung fiehl auf sogenannte JSON Web Tokens, welche seitens des Endpunktes vom DevOps Team konfiguriert wurden. Nach diesem Prinzip wird, sobald bestimmte Nutzende eine Anfrage an OpenSearch unter Verwendung des Dashboards schicken, ein solches Token ausgestellt und hinterlegt, bzw. ein bestehendes Token angegeben, sofern dieses noch gültig ist.

Das finale Endprodukt ist in der Abb. 17 zu sehen.

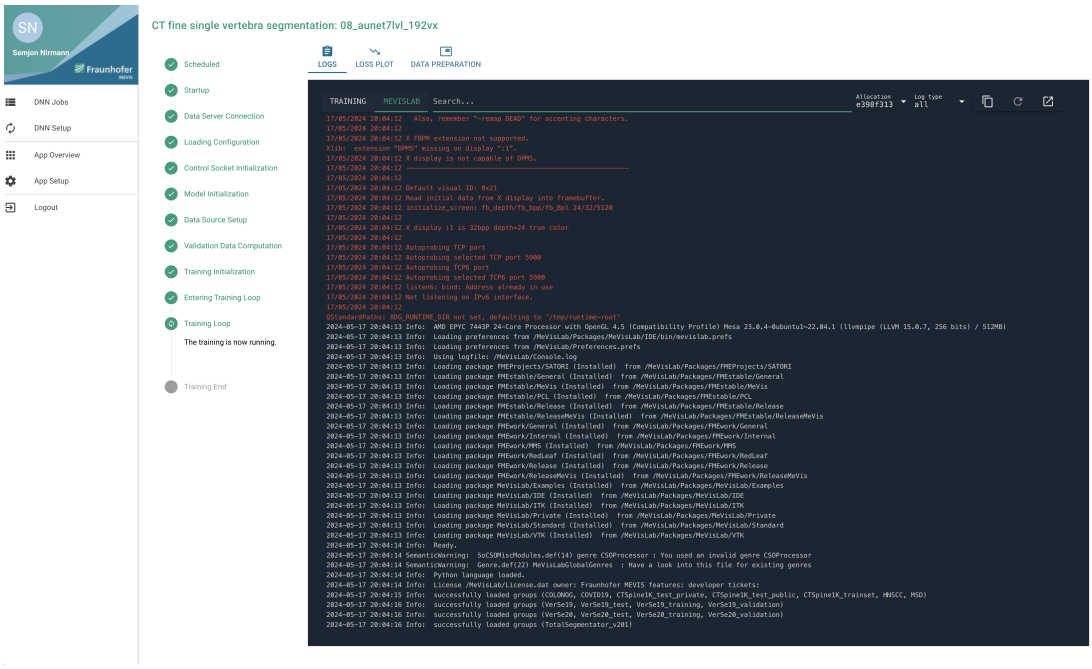


Abbildung 17: Das Endprodukt

Fazit und Ausblick Wir sehen das Projekt "Cluster-Dashboard Logger" als Erfolg an, da diverse Funktionalitäten zuverlässig implementiert werden konnten. Nichtsdestotrotz gibt es Ideen und Ausarbeitungen, die in der Zukunft umgesetzt werden könnten um das Fehlerrisiko zu minimieren und die Wartbarkeit der Anwendung zu verbessern.

- Das Reihenfolgen-Problem lösen: Die Logs der einzelnen Anwendungen selbst müssten ein Format annehmen, in welchem ein Datum mit Millisekundengenauer Uhrzeit enthalten ist. Diese könnte dann von Fluentd geparsed, an Opensearch mitgesendet und dort persistiert werden. Andere Möglichkeiten zur garantierten Reihenfolge gibt es unserer Recherche nach nicht.

- Aktuell werden auch die verschiedenen Sitzungen bzw. Allokationen über OpenSearch eingeholt. Leider ist die Abfrage dieser Sitzungen nicht sortierbar, sodass nicht garantiert werden kann, dass unsere Darstellung der Allokationen (absteigend nach Neuheitsgrad) akkurat ist. Aufgrund diverser Prüfungen, ist die Ausgabe in der Regel korrekt sortiert, es kann allerdings zu Fehlern kommen, sofern nicht die aktuelle Allokation als Erstes vom Endpunkt ausgeliefert wird. Ist das der Fall, so erkennt die Anwendung

nicht, dass sie für das laufende Training den Echtzeit-Modus aktivieren soll bzw. darf. Ein Ausbau der OpenSearch Konfiguration in diesem Punkt wäre dementsprechend förderlich.

- Ausarbeitung der Suchfunktion, sodass reguläre Ausdrücke verwendet werden können, sofern OpenSearch das zulässt

- Modernisierung der Abhängigkeiten und verwendeten Werkzeuge. Dies müsste einheitlich für die gesamte Dashboard-Anwendung geschehen. So sind nicht nur die Versionen der Abhängigkeiten von Belang, sondern auch eine Analyse, welche sich damit beschäftigen sollte, ob es Werkzeuge gibt, welche die entsprechende Aufgabe effizienter oder effektiver erledigen können.

- Die Typsicherheit der Dashboard-Anwendung ist nicht gewährleistet, da hier noch immer JavaScript verwendet wird. Schema-Validierung wäre weiterhin förderlich, bspw. mit der Bibliothek `Zod`.

- Flächendeckende Implementierung von Komponenten- und ggf. End-zu-End-Tests mit Hilfe von Jest bzw. Playwright

- Um den Server weniger zu belasten, wäre es hilfreich statt Polling die WebSocket API zu verwenden.

4.2.2 Blossom

Jessica, Ole, Semjon

Motivation Um die Performanz von neuronalen Netzen gezielt überwachen zu können, können grafische Mittel von großem Nutzen sein. "Blossom" ist der Name einer internen Anwendung, welche im unternehmensinternen Cluster-Dashboard verwendet wird. Diese ist dafür da, den Trainingsverlust sowie weitere Validierungsmetriken des aktuellen Trainings anzuzeigen.

Ein zentraler Aspekt ist die Bereitstellung von Echtzeit-Visualisierungen der Verluste und Metriken während des gesamten Trainingsprozesses. Diese Visualisierungen ermöglichen es, Muster und Trends im Trainingsverlauf unmittelbar zu erkennen und entsprechend darauf reagieren zu können. Durch die Echtzeitüberwachung des Trainingsfortschritts können Benutzer*innen sofortige Einblicke in die Leistung der Modelle erhalten. Dies ermöglicht es Probleme wie Overfitting, Underfitting oder Gradientenexplosionen frühzeitig zu erkennen und gezielt zu adressieren. Blossom wurde gemäß einer Server-Client-Architektur implementiert – der Server empfängt die entsprechenden Trainingsdaten und schreibt sie in einen Redis-Cache, während der Client die persistierten Daten in einem bestimmten Zeitfenster über die GET-Endpunkte des Servers abfragen kann, um diese dann als Graphen darzustellen.

Zum Projektstart war Blossom nicht länger funktionsfähig und wurde nicht mehr gepflegt. Die Fraunhofer MEVIS-Mitarbeiter*innen haben daher auf einen vergleichsweise „unergonomischen“ Workaround wechseln müssen. Bei jedem Validierungsschritt legte ein Trainings-Job auch ein Verlustdiagramm als Bild in PNG-Form in das Ausgabeverzeichnis ab. Dieses Bild konnte als Ersatz für die Grafik im Dashboard näher inspiziert werden.

Ein frühes Ziel war es dementsprechend, Probleme in der Infrastruktur und den CI/CD Pipelines zu identifizieren und zu beheben, um Blossom wieder abrufbar zu machen. Blossom ist eine cloud-basierte Webanwendung, die, wie das Dashboard auch, containerisiert über HashiCorp Nomad bereitgestellt wird. Teil dieser Phase war also die Untersuchung bestehender Pipelines und Konfigurationsdateien für Nomad.

Nach ersten Erfolgen in der Bereitstellung der Anwendung wurde die Zielsetzung weitgehend darauf angepasst, die Effizienz und Benutzbarkeit zu verbessern. Diese Ziele hatten eine Überarbeitung und Modernisierung der Anwendung zur Folge. In diesem Rahmen wurden diverse Abhängigkeiten ausgetauscht, Konfigurationen neu überdacht und anwendungs-interne Routinen hinsichtlich ihrer Laufzeitkomplexität neu evaluiert, da es sich hier um eine clientseitige Anwendung handelt, welche große Datenmengen in Echtzeit verarbeiten muss. Gleichermaßen wurde an Software-Features gearbeitet welche die Interaktivität steigern sollten, wie zum Beispiel das Einstellen der Granularität von Daten und das Abgrenzen verschiedener Trainingssitzungen. Ein zentraler und zeitintensiver Kernaspekt war zuletzt das Smoothing der Daten, da zum Teil viele Tausend Datenpunkte in jedem Training anfallen und visualisiert werden müssen. Mit Smoothing ist in diesem Zusammenhang gemeint, dass die Anzahl der Datenpunkte sinnvoll reduziert wird, um Details auf diese Weise wegzuglätten; so kann die Nutzer*in leichter einen Trend erkennen.

Im Rahmen der Weiterentwicklung von Blossom war das Design einer lokalen Entwicklungsumgebung ein signifikantes Zwischenziel. Diese Umgebung sollte weitesgehend alle Abhängigkeiten zu anderer Software im Cluster wegabstrahieren und gängige Features wie Hot-Module-Replacement, also das Nachladen lokaler Programmcode-Änderungen in die lokal laufende Instanz der Anwendung, bereitstellen. Gleichermaßen soll auch auf Echtzeitdaten der Produktivumgebung zugegriffen werden, ohne diese aber zu verändern. In dieser Phase wurden ebenfalls diverse Komponententests im Blossom-Server implementiert, um die Wartbarkeit und Funktionalität von Blossom zu gewährleisten.

Wiederbeleben Blossom ist eine cloud-basierte Anwendung die über HashiCorp Nomad bereitgestellt wird. HashiCorp Nomad ist eine Anwendung, welche die Orchestrierung von containerisierter Software ermöglicht. In diesem Rahmen werden sogenannte Jobs in Aufgabengruppen unterteilt, sodass jeder solcher Gruppe durch eine zentrale Instanz Ressourcen auf dem Server-Cluster zugewiesen werden können. Gleichermaßen werden unternehmensinterne Bereitstellungen und Änderungen an der Software über GitLab CI/CD Pipelines wirksam gemacht.

Diese Pipelines werden individuell für jede Anwendung konfiguriert, teilen sich aber gewisse Routinen aus bestimmten Repositories des DevOps-Teams "Friday".

Im Rahmen der ersten Phase wurden vorrangig Probleme identifiziert, die verhinderten, dass Blossom schlichtweg neu bereitgestellt werden konnte. Dazu gehörten insbesondere zentrale CI/CD Routinen, welche unter Blossom bislang keine Anwendung fanden. Dadurch waren alte CI/CD Routinen nun unwirksam. Weiterhin wurde die Job-Konfigurationsdatei für Nomad entfernt und musste entsprechend neu geschrieben werden.

Zunächst wurde nach einer Lösung gesucht um die individuellen Komponenten der Anwendung isoliert starten zu können; dafür hat sich Docker angeboten. Mit Hilfe der Konfigurationsdateien der Komponenten wurde außerdem eine docker-compose Konfigurationsdatei erstellt, die deren Zusammenspiel in einem Docker-Netzwerk in einer lokalen Entwicklungsumgebung ermöglicht. Zu guter Letzt konnten die geteilten CI/CD Routinen in Form einer spezifischen CI/CD Pipeline für das Blossom Repository implementiert werden.

Diese Pipeline unterteilt sich in die Phasen `BUILD`, `PUSH` und `DEPLOY`, welche jeweils für den Server und für den Client der Anwendung durchgeführt werden. Dabei werden Docker-Images aus den Konfigurationsdateien gebaut und in der internen Container-Registry veröffentlicht. Der letzte Schritt, `DEPLOY`, findet im Gegensatz zu den anderen Schritten nicht automatisiert statt und wird per Knopfdruck durchgeführt; dieser hat zur Folge, dass der Job auf dem Nomad-Cluster neugestartet wird, sodass die vorangegangenen Änderungen in der Produktivumgebung wirksam werden. Es wurde der Entschluss gefasst, dass diese Pipeline gestartet wird, sobald ein Git Branch in den Master-Branch "gemerged" wird. Es wurde hier also die Entscheidung getroffen, dass mit "Feature Branches" gearbeitet werden soll, welche von dem Master-Branch abzweigen und final wieder hineingezogen werden. Dabei wird zunächst bei Eröffnung von einem Pull-Request die `Build`-Phase durchlaufen, um mögliche Fehler aufzudecken. Läuft diese reibungslos durch, wird der Pull-Request ermöglicht, die Images werden in die Registry geladen, und eine Bereitstellung auf dem Cluster kann durchgeführt werden.

Nachdem diese Pipelines wirksam umgesetzt werden konnten, wurden die Anwendungsabhängigkeiten inspiziert. Der Vulnerability-Scanner der internen Container-Registry (Harbor) hat aufgrund von veralteten Paketen und Container-Images diverse Fehler genannt; so waren beispielsweise die verwendeten Node Versionen veraltet. Diese Abhängigkeiten wurden in dieser Phase ebenfalls auf den neusten Stand aktualisiert.

In der Abbildung 18 erkennt man die alte Version vom Live-Loss Plot. Dieser wird mithilfe der Bibliothek Dygraph in einer Vue-Komponente geladen. Ein großer Nachteil bei der alten Blossom Version bestand darin, dass nicht mit großen Mengen von Datenpunkten umgegangen werden konnte. Gerade bei langen Trainings werden schnell über 100.000 Iterationen erreicht, wobei der Verlustgraph in jeder Iteration einen Datenpunkt für das Trainings-Loss dargestellt hat. Dadurch kommt es zum Einen zu einem sehr langsam, beziehungsweise teilweise gar nicht mehr reaktionsfähigen Graphen und zum Anderen ist es für Benutzer*innen nicht sehr nützlich, jedes einzelne Trainingsloss angezeigt zu bekommen. Dadurch wird die Auswertung von beispielsweise den Validierungsmetriken wie „Accuracy“ oder „Sensitivity“ erschwert, da diese unter den ganzen Datenpunkten des Trainingslosses verschwinden. In der Legende des Graphens tauchen zudem Metriken auf zu denen es keine Datenpunkte gibt.

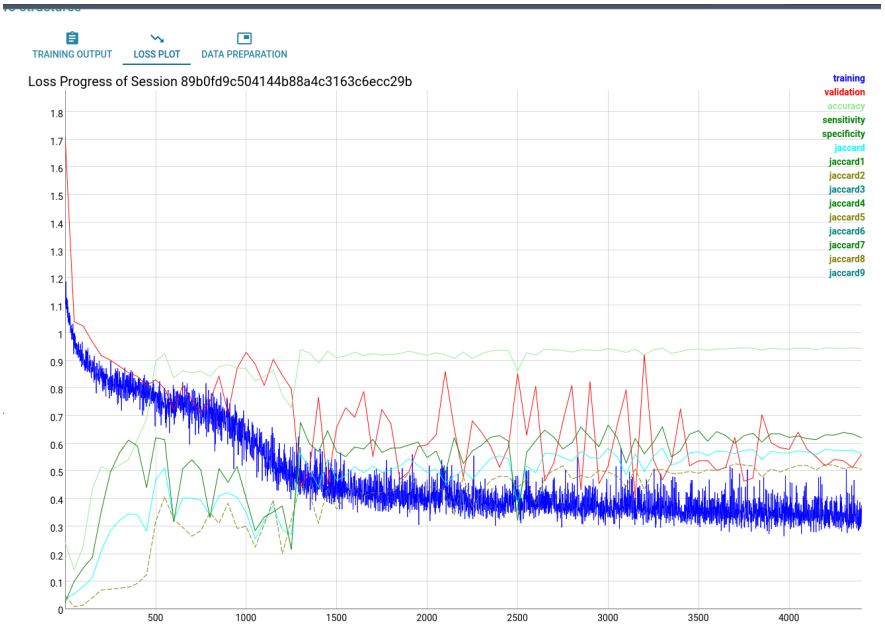


Abbildung 18: Screenshot des alten Live-Loss Plots

Vorbereitungen Um die lokale Entwicklungsumgebung einzurichten wurde das bereits genannte Docker Netzwerk konfiguriert. Dabei wurde die Entscheidung getroffen, alle möglichen Abhängigkeiten zu anderer unternehmensinterner Infrastruktur wegzuabstrahieren; so sollte die Authentisierung und die Beschaffung zentraler Trainingsdaten weiterhin über die Produktivumgebung geregelt werden. Dies wird damit bewerkstelligt, dass diverse Umgebungsvariablen verwendet werden, welche in das Startskript eingespeist werden. Gleichermaßen wird der Redis-Cache auf dem Cluster angesprochen. Die IP und der Port der Allokation auf dem Cluster wird dabei über ein Python-Skript beschafft. Das resultierende Docker-Netzwerk startet letztenendes lediglich das Cluster-Dashboard und den Blossom Server, sowie den Blossom Client lokal und beschafft sich alle notwendigen Daten von Girder (dem Cluster-Dashboard Server) und dem Ticket-Service (einer Abstraktion über dem Vault, welcher für das Lagern von Geheimnissen und das Ausstellen von Authentisierungstokens verantwortlich ist). Die Konfiguration des Docker-Netzwerks verwendet weiterhin sogenannte Volume-Binds, welche lokalen Dateiänderungen lauschen und Änderungen im Blossom Client bzw. Server unmittelbar lokal wirksam machen ohne die Anwendungen neu starten zu müssen.

Verbesserungen Wir haben uns dazu entschieden anstelle der zuvor verwendeten Bibliothek („Dygraph“) eine andere Bibliothek („ChartJS“) zu verwenden. Diese bot uns mehr Freiheit in unserem Vorhaben, einige Verbesserungen in den Plot einzubauen und so die Visualisierung zu modernisieren und zu erweitern.

Der wichtigste Punkt stellte die Implementierung eines Dezimierungsalgorithmus dar, sodass Benuter*innen die Möglichkeit bekommen, die anzuzeigenden Datenpunkte zu reduzieren und lediglich einen Trend, beispielweise des Trainingslosses, anzeigen zu lassen. Um dies während Live-Updates zu ermöglichen und gleichzeitig nicht den Graphen komplett neu rendern zu müssen haben wir die zu reduzierenden Datenpunkten in sogenannten Chunks dargestellt, wobei bei einem Update nur der letzte noch nicht vollständige Chunk aktualisiert wird. Die Chunks selber wurden dann auf die gewünschte Anzahl an reduzierten Datenpunkten dezimiert.

Die Granularität können Benutzer*innen über Slider im Frontend einstellen. Dadurch wurde ermöglicht, dass bei jedem neu reinkommenden Datenpunkt keine

komplette Dezimierung über alle Datenpunkte notwendig war. Dies brachte deutliche Performanzsteigerungen und half dabei, den Graph auch bei großen Datenmengen responsive zu halten.

Durch das Reduzieren der Granularität haben Benutzer*innen die Möglichkeit sich auf Trends in den Metriken zu fokussieren. Zudem können einzelne Metriken über die Legende im oberen Bereich aus- beziehungsweise eingeblendet werden. Hierdurch kann der Fokus auf einzelne Metriken gelegt werden.

Beim Setup eines Trainings werden Parameter, wie zum Beispiel die Lernrate und die Lossfunktion, gesetzt. Diese Parameter entscheiden maßgeblich wie das Modell seine Gewichte, entsprechend des berechneten Verlustes, anpasst und regulieren somit den Trainingsprozess des Neuronalen Netzes. Ein Training kann aus mehreren sogenannten Sessions bestehen. In jeder einer solchen Session wird das selbe Modell trainiert, wobei die oben genannten Parameter allerdings vom vorherigen Training abweichen können. Um Benutzer*innen den Vergleich zwischen unterschiedlichen Sessions zu vereinfachen, werden dessen jeweilige Informationen im Plot dargestellt.

Der Abbildung 19 lassen sich die oben genannten Verbesserung und das Endresultat des neuen Live-Loss Plots entnehmen.

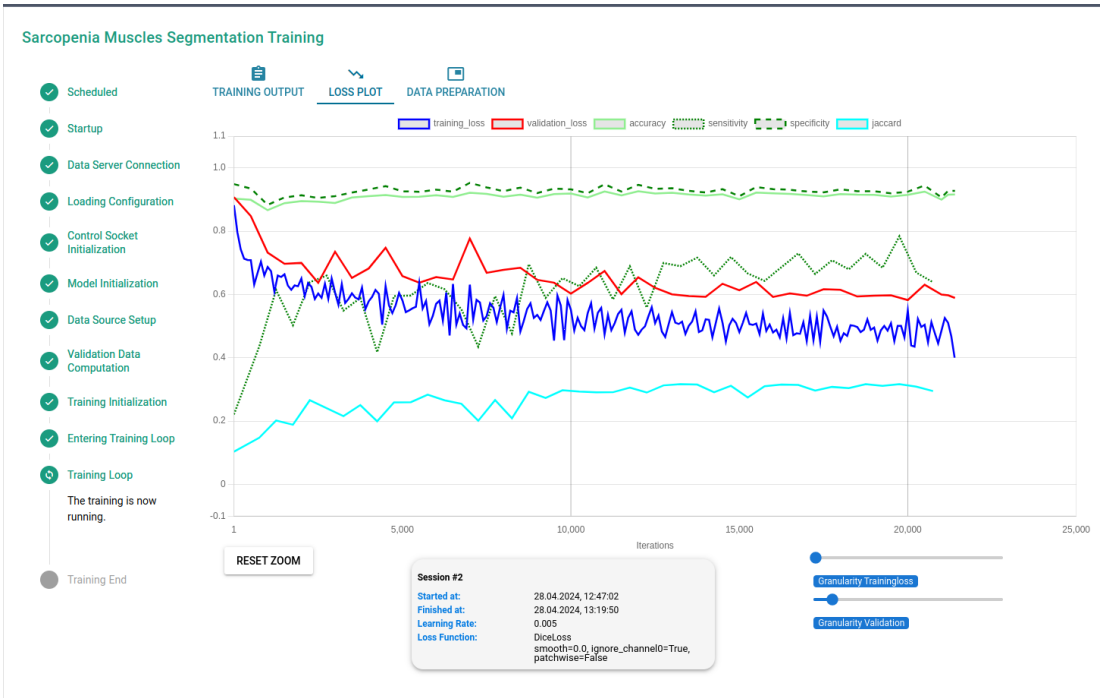


Abbildung 19: Screenshot des neuen Live-Loss Plots

Letzte Optimierungen Nahe dem Projektschluss fand eine Migration auf modernere Werkzeuge statt, um die Wartbarkeit der Anwendung in näherer Zukunft zu gewährleisten.

Der Server wurde dabei von ExpressJS mit einer Node-Laufzeitumgebung auf eine moderne Bun-Laufzeitumgebung umgestellt, unter Verwendung des HonoJS Backend-Frameworks. Hier wurde weitflächig Jest implementiert mit einer Reihe kritischer Redis und API Tests, welche die grundlegende Funktionalität der Persistenzschicht wie auch der Endpunkte prüfen soll.

Der Client wurde auf die modernere Composition-API umgeschrieben, unter Anwendung der neusten Quasar und Vue Versionen.

Sowohl der Client als auch der Server wurden neu mit TypeScript implementiert und verwenden Zod-Validierung; ein Werkzeug, das das ausformulieren von Schemata ermöglicht, welche Ein- und Ausgehende Daten gemäß gewisser Kriterien prüfen. Insgesamt wurde daran gearbeitet, Typsicherheit zu gewährleisten um mögliche Fehler in der Entwicklung bereits in der Phase der Kompilierung zu demaskieren. Auch Redis wurde auf die neuste Version migriert; dabei wurde der veraltete, nicht mehr in Stand gehaltete Redis Client durch den modernen,

offiziellen Redis Client ausgetauscht. Es wurde ebenfalls ESLint für die Prüfung von Code-Stil gemäß allgemein positiv angesehener Praktiken implementiert.

Fazit Das Teilprojekt Blossom kann hinsichtlich der gesetzten Ziele als voller Erfolg angesehen werden. Sämtliche Features funktionieren korrekt, sind bereitgestellt und modernisiert. Benutzer*innen können durch die Echtzeit-Visualisierung der Metriken gestartete Trainings überwachen und entsprechend reagieren falls die gewählten Parameter (wie beispielsweise die Lernrate) zu einem nicht gewollten Verhalten des Trainingsprozesses führen.

Aktuell sind alle für uns sinnvollen und durch Dritte angefragten Funktionen implementiert, eine Erweiterung der Funktionalität kann jederzeit durchgeführt werden. Ziel sollte es im weiteren zeitlichen Verlauf sein, die Integration von Blossom im Dashboard aufrecht zu erhalten und bei Änderungen des Dashboards auch Blossom entsprechend zu aktualisieren.

Die Weiterentwicklung der Anwendung ist vergleichsweise bequem zu bewerkstelligen und die Instandhaltung sollte dadurch gewährleistet sein.

Ausblick Blossom verwendet nun die neusten Werkzeuge auf dem Markt und folgt diversen bewährten Praktiken. Dennoch gibt es weiterhin Optimierungen, deren zukünftige Implementierung lohnenswert wäre, da es sich um eine komplexe Anwendung handelt:

- Middleware
  - Im Server wurde Middleware sparsam eingesetzt und deckt die grundsätzlichen Anwendungsfälle von Cross Origin Resource Sharing und Logging ab. Einheitliche Middleware für die Fehlerbehandlung und detaillierteres Logging wären wünschenswert.
- Tests
  - Die implementierten Tests stellen das absolute Minimum dar. Diverse Fehlerfälle werden noch nicht in den Tests geprüft.
- Typsicherheit

- End-zu-End Typsicherheit ist nicht gewährleistet. Aktuell ist die korrekte Funktionalität im Client davon Abhängig, dass Änderungen der Typ-Spezifikationen aus dem Server in den Client übertragen werden. Werkzeuge wie RPC stellen vielversprechende Mittel dar um, typsichere API-Endpunkte zu gewährleisten, dafür wären allerdings Umstellungen des Blossom-Designs notwendig. So müsste der Client gebaut und statisch über den Server ausgeliefert werden, statt den Client und den Server als getrennte Anwendungen zu betrachten.

- Ein Repository für die Typ-Spezifikationen wäre weiterhin hilfreich, sodass der Client und der Server dieses einfach implementieren könnten.

- Es wäre förderlich, wenn Änderungen am Blossom-Logger in RedLeaf automatisch dazu führen würden, dass diese Typspezifikationen angepasst würden. Dafür sind Werkzeuge wie Swagger oder OpenAPI interessant.

5 Modellevaluation: Muskelsegmentierung

Ardit

Motivation Einer der Gründe für die Durchführung der Modellevaluation war die Bewertung der Effizienz und Genauigkeit der Segmentierungsmodelle, sowie die Schaffung einer Plattform für zukünftige Verbesserungen durch Tests und Vergleiche. Mit einer Vielzahl von Segmentierungsmethoden ist der Bedarf an Evaluierung gewachsen, hier vergleichen wir zwei verschiedene Muskelsegmentierungsmethoden, um ihre Unterschiede, Stärken und Schwächen zu entdecken.

Eines der verfügbaren Modelle wird von DIAG (Diagnostic Image Analysis Group) mit Sitz in Nijmegen, Holland, bereitgestellt. Es ist in der Lage, die gesamte Körperlänge zu segmentieren und zwischen Skelettmuskeln und Körperfett zu unterscheiden, welches in drei verschiedene Typen unterteilt wird: subkutanes Fett, intramuskuläres Fett und viszerales Fett. Anstelle des Algorithmus wurden uns markierte Daten aus einer gemeinsamen Datenbank mit tatsächlichen CT-Scans von Menschen zur Verfügung gestellt. Die Daten wurden im Nifti-Format geliefert, wobei jede Datei von 30 bis 800 Schnitte des menschlichen Torsos enthält. Es ist erwähnenswert, dass der Modell an sich keine Unterschiede zwischen verschiedenen Muskelgruppen wie Rücken- oder Bauchmuskeln markiert, sondern sie als eine einzige Struktur erkennt.

Das zweite für uns verfügbare Modell heißt TotalSegmentator und ist ein quelloffenes Programm, welches in der Bildverarbeitung verwendet wird und insgesamt 117 verschiedene Klassen segmentieren kann. Unser Fokus umfasst nur eine dieser Klassen, die als "Tissue-Types" bezeichnet werden, die ebenfalls die gesamte Körperlänge segmentieren und zwischen Skelettmuskeln, sowie nur zwei Arten von Körperfett unterscheiden kann, dem subkutanen und viszeralen Fett.

Das von uns trainierte Modell ist darauf ausgelegt, Bauch und Rückenmuskeln als getrennte Strukturen auf der Höhe der Lendenwirbel zwei bis vier zu erkennen. Intramuskuläres Fett wird nicht als separate Struktur erkannt, es wird aber auch nicht zu den Bauch- oder Rückenmuskeln hinzugezählt.

Nach Bestätigung der Ressourcen bestand der nächste Schritt darin, ein konkretes und erreichbares Ziel festzulegen und dann eine Infrastruktur zu entwerfen, die mit den uns zur Verfügung gestellten Werkzeugen dieses Ziel erreichen kann. Referenzdaten sind nicht vorhanden, da diese nur nach einem zeitaufwändigen Prozess bereitgestellt werden können, bei dem die Daten von einem autorisierten Radiologen von Hand gelabelt werden. Hierdurch ist es jedoch nicht möglich, zu einem qualifizierten Schluss zu kommen, welches Segmentierungsmodell besser ist als das andere. Es ist jedoch möglich, durch visuelle und quantitative Vergleiche die Unterschiede und Ähnlichkeiten zwischen allen Modellen festzustellen.

Zielsetzung Das Setzen von Zielen und Erwartungen ist ein notwendiger Schritt, um sicherzustellen, dass der Umfang des Projekts nicht überschritten wird und wir spezifische Ergebnisse termingerecht liefern können. Es ist wichtig zu wissen, dass unser Vergleich streng auf den Muskelbereich einschließlich des L1-L4 Bereichs beschränkt ist, der ein wichtiger Indikator für Sarkopenie ist, da nur eines der Modelle intramuskuläres Fett unterscheiden kann.

Wir wurden auch mit einem Szenario konfrontiert, bei dem wir keine Referenzdaten hatten, um unsere Ergebnisse zu vergleichen, und darüber hinaus auch kein Zugriff auf den DIAG-Algorithmus bestand. Dies veranlasste einen ergebnisbasierten Vergleich aus zwei verschiedenen Perspektiven:

- Visuell basierter Vergleich
  Nach der Segmentierung der Rohdaten mit unserem Deep Learning Modell und dem TotalSegmentator-Algorithmus war es möglich, die Ergebnisse zu visualisieren und sie zur klaren Vergleichbarkeit auf die Originaldaten zu überlagern. Die Ansicht wurde uns dann in 2D mit einem verbundenen 3D-Modul, das den aktuellen Standort der angezeigten Ansicht repräsentiert, dargestellt. Durch Scrollen war es möglich, zu verschiedenen Schnitten des Torsos zu wechseln und eine vollständige Ansicht der Leistung der Algorithmen zu erhalten.

  Wie im Abbildung 21 zu sehen ist, ist die Maske des DIAG-Algorithmus in blau und die des TotalSegmentator-Algorithmus in rot markiert. Wir bevorzugen es, Farben zu verwenden, um zwischen den beiden Ergebnissen

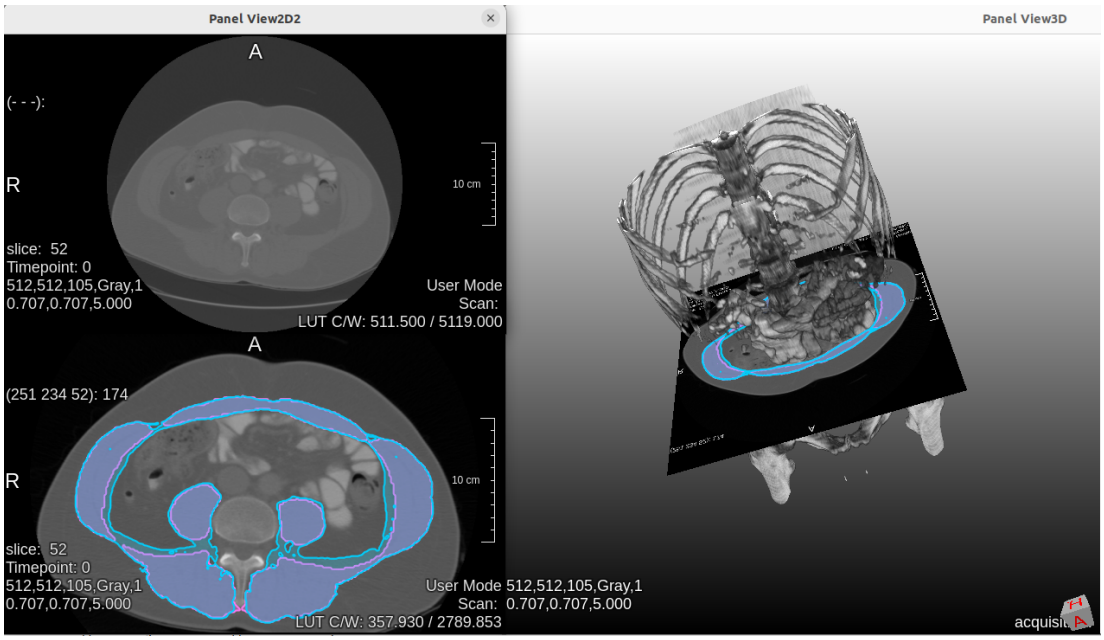


Abbildung 20: Screenshot der 3D-Visualisierung

zu unterscheiden. Diese Methode erwies sich als sehr nützlich, um die beiden Leistungen visuell zu vergleichen, aber es gab Probleme, die Bereiche zu identifizieren, die nur von einem Abschnitt annotiert wurden, insbesondere bei der Durchsicht von 300 Schnitten. Um diesen Prozess effektiver zu gestalten, beschlossen wir, einen Schritt weiter zu gehen und eine Ansicht beider Masken ohne die gemeinsam annotierten Bereiche einzubeziehen. Dies ermöglichte es uns, die Konturen und Liniendefinition besser zu erkennen und unsere Wahrnehmung zu verbessern, um die Unterschiede zwischen den beiden Ergebnissen zu deutlicher aufzuzeigen.

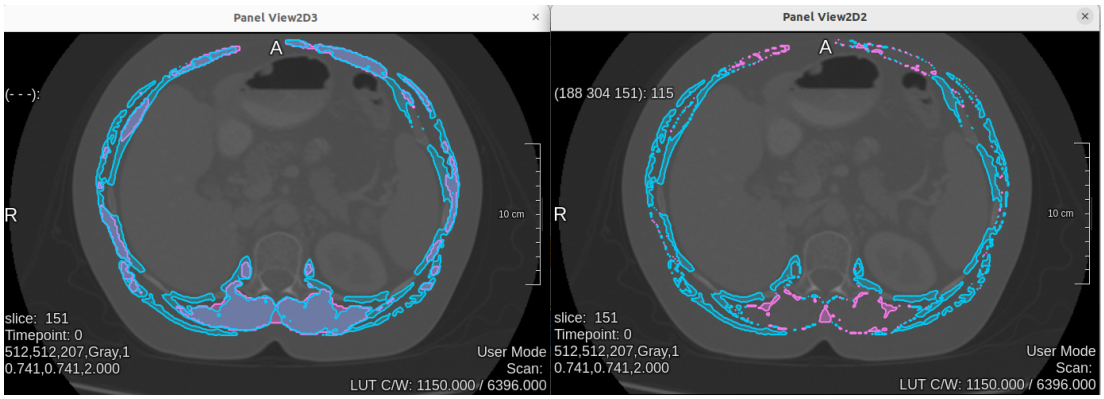


Abbildung 21: Screenshot der überlappenden Masken

- Metrikbasierter Vergleich

Die Verwendung verschiedener Metriken zur Bewertung unserer Modelle kann dazu beitragen, die Funktionsweise der Algorithmen besser zu verstehen. Es ist eine ergänzende Methode, die besser unterstützt, Statistiken wie Präzision, Volumen sowie Weitere zu identifizieren. Für einen besseren Überblick haben wir beschlossen, eine Tabelle mit vier der Metriken zu erstellen:

- Dice-Koeffizient:
  Der Dice-Koeffizient misst die pixelweise Übereinstimmung zwischen einer Segmentierung und der entsprechenden Grundwahrheit. Der Wert kann zwischen null und eins variieren, wobei eins eine perfekte Übereinstimmung bedeutet. Hier wird er nur verwendet, um die Überlappung zwischen zwei verschiedenen Masken zu messen und liefert keine Informationen darüber, ob diese Überlappung korrekt ist oder nicht. Er zeigt lediglich den Grad an, zu dem sich diese Masken ähneln.

- Volumetric Difference:
  Die Volumetric-Difference misst die unterschiedlichen Volumina jeder der Masken und liefert die Differenz zwischen den beiden. Sie ist notwendig, um zu quantifizieren, wie stark sich die Volumina der segmentierten Bereiche unterscheiden.

- Maximale Hausdorff-Distance:
  Die maximale Hausdorff-Distanz misst die größte Entfernung zwischen einem Punkt in einer Maske und dem nächstgelegenen Punkt in einer anderen Maske. Sie ist notwendig, um die äußerste Diskrepanz zwischen den beiden Masken zu identifizieren und zu quantifizieren.

- Mean Distance:
  Die Mean Distanz misst die durchschnittliche Entfernung zwischen entsprechenden Punkten in zwei verschiedenen Masken. Sie ist notwendig, um den allgemeinen Grad der Übereinstimmung und die Genauigkeit der Segmentierung zu bewerten.

Datenverarbeitung Nach den anfänglichen Vorbereitungen bestand der nächste Schritt darin, die benötigten Daten von anderen Algorithmen zu sammeln und die erforderliche Architektur aufzubauen, um die Ergebnisse mithilfe von MeVisLab zu vergleichen. Von den Rohdaten, die 300 verschiedene Fälle umfassen, bestanden die von DIAG bereitgestellten Ergebnisse aus 123 zufällig aus diesen 300 ausgewählten Fällen. Dieser Teil stellt den gesamten Datensatz dar, der für diese Analyse verwendet wird.

Nach der Installation der TotalSegmentator-Software und dem Erwerb der entsprechenden Lizenz wurde das Programm lokal ausgeführt, und jeder der 123 Fälle wurde manuell beschriftet. Anschließend wurde ein neues Verzeichnis in Gaia erstellt, um die Dateien zu speichern.

Unser Deep Learning Team lieferte Ergebnisse aus ihrem eigenen Modell, das Fälle von 0 bis 100 umfasste und etwa 40 Prozent aller Fälle repräsentierte. Aufgrund von Zeitbeschränkungen wurde festgestellt, dass diese Stichprobengröße ausreicht, um eine genaue Bewertung durchzuführen.

Mit den beschafften Daten bestand die nächste Priorität darin, eine geeignete Architektur zu entwerfen. Das Modell musste folgende Aufgaben erfüllen können:

- Alle Daten in MeVisLab lesen und laden:
  Es war notwendig, mindestens zwei überlappende Masken zusammen mit den Rohdaten zu verwenden, um einen genauen Vergleich zu ermöglichen. Eine .lst-Datei wurde erstellt, um die Verzeichnisse aller Rohdateien zu enthalten, und die anderen beiden Verzeichnisse wurden in das Modul in MeVisLab geladen. Mithilfe der Weltmatrix stellten wir sicher, dass die Masken gut positioniert waren.

- Unterscheidung zwischen Skelettmuskel und Fett:
  Dies wurde durch die Einrichtung eines Threshold erreicht, um Muskel von den anderen Labels zu trennen. Das DIAG-Modell hatte vier verschiedene Threshold-Werte verfügbar: einen für Skelettmuskel und drei für verschiedene Arten von Fett, während das TotalSegmentator-Modell nur drei Threshold-Werte hatte, da es kein intramuskuläres Fett erkennen konnte.

- Informationen visuell darstellen:
  MeVisLab ist mit sowohl 2D- als auch 3D-Viewer-Modulen ausgestattet, die leicht verbunden werden können, um die gewünschten Informationen anzuzeigen. Wir richteten insgesamt vier 2D-Viewer ein, die entlang der Längsachse betrieben wurden: zwei, um eine vollständige Ansicht der Modelle separat darzustellen, die alle Labels enthalten, bevor der Threshold-Wert angewendet wurde, und die anderen beiden, um die überlappenden Masken in verschiedenen Farben anzuzeigen, wie im Abschnitt Zielsetzung beschrieben. Darüber hinaus fügten wir ein 3D-Viewer-Modul hinzu, das mit dem 2D-Overlap-Viewer verbunden ist, um den Standort der aktuellen Masken anzuzeigen.

- Die gewünschten Metriken berechnen:
  Dies wurde durch bestimmte MeVisLab-Module wie CompareMatrix und ComputeConfusionMatrix ermöglicht, welche automatisch verschiedene Metriken berechnen, die für jede Berechnung erforderlich sind.

Ergebnisse Nach der Analyse der einzelnen Fälle haben wir eine Ordnerstruktur organisiert, in der jeder Fall einen Unterordner mit einem Screenshot der notwendigen Metriken, wie in Abbildung 23 gezeigt, sowie zwei weiteren Screenshots zur visuellen Darstellung des Modellvergleichs enthält. Die Metriken werden dann in eine Tabelle formatiert, um eine bessere Ansicht und zukünftige Referenz zu ermöglichen. Alle Daten, zusammen mit der zum Modellvergleich verwendeten Architektur, werden in Gaia hochgeladen, damit zukünftige Teilnehmer des DeepAnatomy Projekts darauf zugreifen können.

Aus einer allgemeinen Sichtweise können wir nach mehreren Beobachtungen feststellen, dass alle drei Modelle in den meisten Fällen den Bereich der Skelettmuskulatur klassifizieren können. Leider können wir die Genauigkeit dieser Vorhersagen nicht bestimmen. Nach der Analyse der Ergebnisse sind jedoch unsere Erkenntnisse wie folgt:

- Das DIAG Modell

  Im Vergleich zu den anderen beiden Modellen neigt das Modell von DIAG dazu, eine umfangreichere Flächenabdeckung zu haben, was darauf hindeuten könnte, dass es zusätzliches Muskelgewebe oder möglicherweise einige nicht-Muskelbereiche einschließt. Dies betrifft sowohl die Rückenmuskulatur als auch die Bauchmuskulatur. Im Vergleich zu TotalSegmentator und Deep Learning Modellen beträgt der durchschnittliche Volumenunterschied $-417,90$ ml bzw. $-2696,47$ ml, was eine höhere Abdeckung beweist. Trotzdem hatte es eine gute Überlappung mit TotalSegmentator mit einem durchschnittlichen Dice-Koeffizientenwert von $0,89$. Diese beiden Faktoren erklären, warum die meisten der äußeren Konturen, die in den überlappenden Modellen beobachtet wurden, zu DIAG gehörten. Mithilfe der distanzbasierten Metriken konnten wir die mittlere Distanz zwischen den Konturen beider Modelle bestimmen, was zu einem durchschnittlichen Wert von $1,43$ mm führte.

  Es gab jedoch auch Unstimmigkeiten und Abweichungen, die hauptsächlich in der unteren Hälfte des Torsos auffielen. Wir verwendeten die maximale Hausdorff-Distanz, um solche Abweichungen zu berechnen, was zu einem durchschnittlichen Wert von $37,28$ mm und einem Höchstwert von $92,59$ mm führte. Dies allein beweist, dass in einigen Bereichen starke Unterschiede zwischen den beiden Modellen bestehen. In der oberen Hälfte des Torsos gibt es eine einheitlichere Ansicht, bei der es scheint, dass beide Modelle weniger Meinungsverschiedenheiten haben.

  Darüber hinaus ist es erwähnenswert, dass in einigen speziellen Fällen, wie Fall Nummer 115, bei denen Fremdkörper wie Schrauben oder metallische Komponenten in den Körper eingebracht wurden, sich die Vorhersagen des Modells erheblich änderten. Dies wurde auch in zwei anderen Fällen beobachtet, was zu der Annahme führt, dass das Modell nicht gut reagiert, wenn das Muster gestört wird. Außerdem enthält die Abdeckung oft Löcher und Unterbrechungen unterschiedlicher Größe, die, wenn sie in den Originaldaten ohne aktivierten Schwellenwert analysiert werden, als intramuskuläres Fett erkannt werden können, das sich innerhalb oder an der Seite der Skelettmuskulatur befindet.

- Das TotalSegmentator Modell

  Im Vergleich zu den anderen beiden Modellen hat das Modell von TotalSegmentator ein deutlicheres und vollständigeres Erscheinungsbild, was darauf zurückzuführen ist, dass es intramuskuläres Fett nicht erkennt, sondern stattdessen in den Muskel einschließt. Es hat eine konservativere Abdeckung, was zu einer unidentifizierten Muskelmasse führen könnte, insbesondere wenn die Genauigkeit der Muskelmasse ein wichtiger Faktor bei der Erkennung von Sarkopenie ist. Es hat eine gute Überlappungskompatibilität mit dem Modell von DIAG, da es dazu neigt, unterzusegmentieren, aber es gibt Bereiche mit großen Abweichungen. Dies hat sich manchmal als wertvoller Unterschied erwiesen, da das Modell von TotalSegmentator in der Lage war, den Mutterreibe-Muskel in den Fällen 39, 40 und 72 zu erkennen und zu kennzeichnen, was den anderen beiden Modellen zuvor entgangen war. Gleichzeitig können diese Abweichungen auch Fehler sein, die durch die Kennzeichnung von falschem Gewebe innerhalb des subkutanen Fettbereichs verursacht werden, wie in Fall 64 bei Abbildung 22 zu sehen ist.

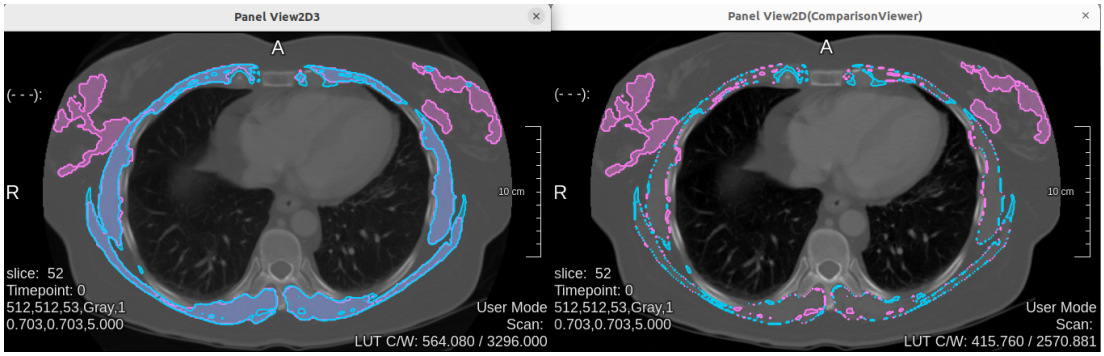


Abbildung 22: TotalSegmentator Anomalie der Muskelsegmentierung in Fall Nr. 64

- Deep Learning Modell

  Es war schwierig, dieses Modell mit den anderen beiden zu bewerten, da seine Leistung hauptsächlich auf den Bereich L1-L4 beschränkt ist. Aus diesem Grund zeigen die berechneten Metriken einen signifikanten Unterschied im Vergleich zu den anderen Modellen, da sie die Leistung auf dem gesamten Torso und nicht in einem bestimmten Bereich berechnen. Im Vergleich zu DIAGs Modell ergab sich ein Dice-Koeffizientenwert, der von 0,39 bis 0,85 variierte, mit einem durchschnittlichen Wert von 0,65. Wenn korrekt

interpretiert, bedeutet dies nicht unbedingt, dass seine Leistung nicht den Anforderungen entspricht, sondern vielmehr, dass sein Volumen außerhalb des L1-L4-Bereichs gering ist, was zu einem kleinen Überlappungswert führt.

Nach gründlichen visuellen Beobachtungen stellten wir fest, dass dieses Modell in der Lage war, die kleine Verbindung zwischen den Rückenmuskeln zu kennzeichnen, die nur von TotalSegmentator modellfallweise erkennbar blieb. Darüber hinaus ist seine Struktur auch voller Ausschlüsse, was zu der Erkenntnis führt, dass es intramuskuläres Fett aus der Struktur ausschließen kann. Auf der negativen Seite traten bestimmte Anomalien auf, bei denen die Bauchmuskeln nur teilweise erkannt wurden, wie in Fall 12, oder wenn ein kleiner Teil des Organgewebes ebenfalls als Muskel klassifiziert wurde. Es ist bemerkenswert, dass seine Leistung, wenn sie visuell mit den anderen Modellen im L1-L4-Bereich verglichen wurde, angemessen war.

Zusammenfassend hat jedes Modell seine eigenen Besonderheiten und Ähnlichkeiten zu anderen, was zu erwarten ist, da sie dieselben Körperteile segmentieren. Es war interessant zu sehen, wie sie im Vergleich zueinander abschneiden. Wenn mehr Zeit zur Verfügung stünde, würden wir sie mit einem größeren Datensatz mit mehr Variablen weiter testen. Ein nächster Schritt wäre auch, eine Bounding-Box einzuschließen und die Bewertung nur in bestimmten Regionen wie L1-L4 durchzuführen, um bessere Ergebnisse zu erzielen. Leider war dies aufgrund von Zeitbeschränkungen nicht möglich.

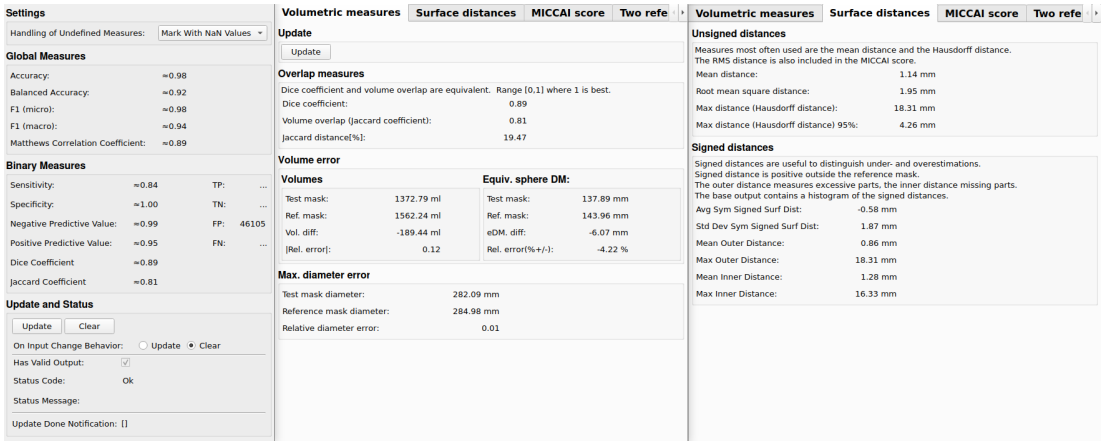


Abbildung 23: Metrikergebnisse aus Fall Nr. 42 TotalSegmentator-DIAG

6 Fazit und Ausblick

Jorma, Semjon

Das Software-"Ökosystem" vom Fraunhofer MEVIS ist vielfältig und komplex. Im Laufe dieses Projekts standen wir vor diversen Herausforderungen uns in die cloud-basierte Infrastruktur einzuarbeiten, verwende Dienste verstehen und erweitern zu können, sowie uns eine grundlegende fachliche Expertise im Rahmen der künstlichen Intelligenz anzueignen. Nicht zu letzt stellte das organisatorische Umfeld und die Einarbeitung in das Team eine Herausforderung dar, welche gut gemeistert werden konnte, durch die zuvorkommende Hilfe verschiedener Mitarbeiter und Abteilungen. Trotz anfänglicher Schwierigkeiten durch die neue Arbeitsumgebung beim Fraunhofer MEVIS haben wir als Team gemeinsam eine erfolgreiche Basis aufgebaut und erweitert, die nicht nur von den Mitarbeitern bei MEVIS in der Zukunft genutzt werden kann, sondern auch nachfolgenden Teilnehmern des DeepAnatomy-Projektes als Grundlage dienen könnte.

Wir haben das Ziel erreicht, für Sarkopenie ein Model zu trainieren, das Bauch- und Rückenmuskeln erkennen kann. Obwohl das erzielte Ergebnis bereits funktioniert, könnte die Zuverlässigkeit noch weiter gesteigert werden. Mit mehr Zeit für weitere Experimente und intensiverem Training könnten die Ergebnisse dieses Modells weiter verbessert werden.

Das Cluster-Dashboard wurde auf diverse Arten erweitert. Unter Betrachtung eines bestimmten Trainings funktionierten so zwei von drei Hauptfunktionalitäten zu Projektbeginn gar nicht; es konnte weder die Trainingsentwicklung inspiziert, noch Statusmeldungen zum Trainingsverlauf betrachtet werden. Beide dieser Funktionen wurden durch Blossom und den Dashboard-Logger erbaut, verbessert und bereitgestellt. Die dafür angefertigte Software ist ausführlich dokumentiert und verwendet moderne Werkzeuge, die die Wartbarkeit und Qualität sicherstellen sollen. Es wurden weiterhin Pipelines implementiert, die die Bereitstellung von Änderungen vereinfachen, sowie minimale lokale Entwicklungsumgebungen eingerichtet die eine effiziente Bearbeitung der Anwendungen ermöglichen.

Es wurden im Rahmen des Cluster-Dashboards weiterhin erfolgreich neue Inputs zum Eingeben von GitLab-Links und Tokens zum herunterladen von Dateien des

zugehörigen Repositories eingefügt, jedoch steht deren Funktionalität noch in Frage. Somit kann dieses Projekt nicht als abgeschlossen betrachtet werden.

Wir sind zuversichtlich, dass die kommenden DeepAnatomy-Teilnehmer**innen* auf unserer Arbeit aufbauen können sowie die entwickelten Tools verbessern werden. Wir hoffen weiterhin, dass dieser Projektbericht einen Einblick in die zugrundeliegenden Probleme und die Motivation für bestimmte Entscheidungen bieten kann. Des Weiteren bieten die Sektionen, in denen ein jeweiliger Ausblick zu dem entsprechenden Teilprojekt geboten wird, Ansätze und Werkzeuge, die auf interessante Art und Weise in die Anwendungslandschaft einfließen könnten.

Zuletzt blicken wir optimistisch auf die zukünftige Entwicklung von DeepAnatomy und freuen uns daürber, für die zunehmend an Bedeutung gewinnende medizinische Bildverarbeitung einen Beitrag geleistet zu haben.

7 Danksagung

Projektteam

Unser besonderer Dank gilt Dr. Hans Meine und Felix Thielke für ihre umfangreiche Betreuung und Unterstützung während des gesamten Projekts. Ebenso möchten wir allen Kollegen und Kolleginnen bei MEVIS danken, die uns stets mit Rat und Tat zur Seite standen. Ihre Hilfsbereitschaft und ihr Fachwissen haben uns in vielen Situationen weitergeholfen und das Projekt bereichert.

A Anhang

Literatur

Behboodi, B., Fortin, M., Belasso, C. J., Brooks, R., & Rivaz, H. (2020). Receptive field size as a key design parameter for ultrasound image segmentation with u-net. *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, 2117–2120.

Camgözlü, Y., & Kutlu, Y. (2020). Analysis of filter size effect in deep learning. *arXiv preprint arXiv:2101.01115*.

Cruz-Jentoft, A. J., Bahat, G., Bauer, J., Boirie, Y., Bruyère, O., Cederholm, T., Cooper, C., Landi, F., Rolland, Y., Sayer, A. A., et al. (2019). Sarcopenia: revised European consensus on definition and diagnosis. *Age and ageing*, *48*(1), 16–31.

Du, G., Cao, X., Liang, J., Chen, X., & Zhan, Y. (2020). Medical Image Segmentation based on U-Net: A Review. *Journal of Imaging Science & Technology*, *64*(2).

getbem. (2024). BEM – Naming [https://getbem.com/naming/ [Accessed: 19.05.2024]].

Grafana Labs. (2024). Grafana Loki Documentation | Promtail [https://grafana.com/docs/loki/latest/send-data/promtail/ [Accessed: 19.05.2024]].

HashiCorp, I. (2024a). Create a cluster | Nomad | HashiCorp Developer [https://developer.hashicorp.com/nomad/tutorials/get-started/gs-start-a-cluster [Accessed: 19.05.2024]].

HashiCorp, I. (2024b). Filesystem | Nomad | HashiCorp Developer [https://developer.hashicorp.com/nomad/docs/concepts/filesystem [Accessed: 19.05.2024]].

HashiCorp, I. (2024c). Glossary | Nomad | HashiCorp Developer [https://developer.hashicorp.com/nomad/docs/glossary#job [Accessed: 19.05.2024]].

Hurezeanu2, A. D. A. R. (2023). Sarcopenia. *StatPearls*. https://www.ncbi.nlm.nih.gov/books/NBK560813/

Jena, B., Jain, S., Nayak, G. K., & Saxena, S. (2023). Analysis of depth variation of U-NET architecture for brain tumor segmentation. *Multimedia Tools and Applications*, *82*(7), 10723–10743.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436–444.

Mingrammer. (2024). GitHub: Mingrammer/Flog [https : / / github . com / mingrammer/flog [Accessed: 19.05.2024]].

Mourtzakis, M., Prado, C. M., Lieffers, J. R., Reiman, T., McCargar, L. J., & Baracos, V. E. (2008). A practical and precise approach to quantification of body composition in cancer patients using computed tomography images acquired during routine care. *Applied Physiology, Nutrition, and Metabolism*, *33*(5), 997–1006.

Shen, D., Wu, G., & Suk, H.-I. (2017). Deep learning in medical image analysis. *Annual review of biomedical engineering*, *19*, 221–248.

Zhang, Y., Liu, S., Li, C., & Wang, J. (2021). Rethinking the dice loss for deep learning lesion segmentation in medical images. *Journal of Shanghai Jiaotong University (Science)*, *26*, 93–102.