Universität Bremen

Fraunhofer MEVIS

Master-Projekt DeepAnatomy

Projektbericht

Teilnehmer:

Jannes Adam	jadam@uni-bremen.de
Niklas Agethen	agethen@uni-bremen.de
Robert Bohnsack	bohnsack@uni-bremen.de
René Finzel	rfinzel@uni-bremen.de
Timo Günnemann	timo2@uni-bremen.de
Lena Philipp	len_phi@uni-bremen.de
Marcel Plutat	mplutat@uni-bremen.de
Markus Rink	marink@uni-bremen.de
Tingting Xue	xue@uni-bremen.de

Betreut durch:

Hans Meine	hans.meine@mevis.fraunhofer.de
Felix Thielke	felix.thielke@mevis.fraunhofer.de

8. Oktober 2021

AG Medical Image Computing

Universität Bremen

WiSe 20/21 & SoSe 21

Inhaltsverzeichnis

1 Einleitung

1.1 Ziel des Projekts

AUTOR*IN: TINGTING XUE

Dieses Masterprojekt zielt darauf ab, eine einfach zu bedienende Plattform für die medizinische Bildsegmentierung zu entwickeln, mit der Segmentierungsmodelle durch Deep Learning trainiert werden können. Forscher*innen können diese Plattform verwenden, um viele verschiedene Segmentierungsaufgaben zu bewältigen.
Zu diesem Zweck konzentrieren wir uns auf folgende Entwicklung und Forschung:

- Entwicklung und Implementierung verschiedener, bestehender Bildsegmentierungsalgorithmen und -methoden (wie U-Net usw.) in der Plattform
- Bereitstellung einer Plattform, die für den Vergleich der Leistungs- und Segmentierungsergebnisse verschiedener Algorithmen für weitere Analysen und Forschungen geeignet ist
- Erweiterung der Funktionen der MeVisLab-Software, z.B. die Verwendung vorhandener Module zum Aufbau einer Struktur zur Lösung neuer Aufgaben
- Verwendung einer großen Menge an Daten und Experimenten, um die Richtigkeit und Verwendbarkeit der Entwicklung zu überprüfen, und Recherche der optimalen Algorithmen und Parameter für bestimmte Aufgabenhintergründe (wie z.B. Nieren)

- Verbesserung unserer persönlichen Entwicklungsfähigkeit und Teamfähigkeit
- Vertraut machen mit verschiedenen Kollaborationsplattformen und Online-Arbeitsmethoden in der Teamzusammenarbeit
- Einreichung der Segmentierungsergebnisse bei einem international renommierten Wettbewerb um mit Teams aus verschiedenen Ländern zu konkurrieren

Wir erwarten, dass unsere Erweiterungen die Anwendbarkeit von MeVisLab in relevanten Bereichen weiter fördern.

1.2 KiTS-Challenge

AUTOR*IN: NIKLAS AGETHEN UND JANNES ADAM

Da wir zu Beginn des Projekts eher wenig Erfahrung mit Bildverarbeitung im medizinischen Kontext hatten, nutzten wir die Kidney Tumor Segmentation (KiTS)-Challenge, die 2019 stattfand, um Kenntnisse zu sammeln, die MEVIS-Infrastruktur kennen zu lernen und konkrete Aufgaben für unser Projekt entwickeln zu können.
Die KiTS-Challenge hat das Ziel anatomische Strukturen in der Nierenregion zu segmentieren und wird in einer Kollaboration zwischen dem *University of Minnesota Robotics Institute*, der *Helmholtz Imaging Platform* und dem *Cleveland Clinic's Urologic Cancer Program* organisiert.
Während unser Projekt lief, startete die KiTS21-Challenge, an der wir teilnahmen. 2019 wurde bei der Premiere der Fokus auf die Segmentierung von Nierentumoren gelegt. 2021 kamen zusätzlich Nierenzysten hinzu, die segmentiert werden sollten, da diese nach operativer Tumorentfernung zur Beurteilung der Nierenfunktion medizinisch relevant sind.
Für die Teilnahme an der Challenge stellen die Organisatoren CT-Bilder und Annotationen zur Verfügung. Die teilnehmenden Teams geben ein End-to-End-System in Form eines Docker-Containers ab, das die erwähnten Strukturen bestmöglich segmentiert. Außerdem muss das System in einem Short-Paper beschrieben werden.

Die Abgaben werden durch die Organisatoren anhand eines vordefinierter Metriken und eines separaten, nicht öffentlichen Datensatzes verglichen und die besten Lösungen prämiert. Außerdem gibt es die Möglichkeit das Paper bei dem Satellite Event bei der MICCAI 2021, der International Conference on Medical Image Computing and Computer Assisted Intervention, zu präsentieren [1].

2 Organisation

2.1 Projektmanagement

2.1.1 Workflow

AUTOR*IN: ROBERT BOHNSACK

Um den Fortschritt von Aufgaben im Blick zu behalten und zukünftige Aufgaben zu planen und festzuhalten wurden im Projekt Jira und Git genutzt.

2.1.1.1 Jira

Jira[1] ist eine von Atlassian entwickelte Projektmanagement Software. Diese erlaubt es mit Kanban Boards und weiteren Features, Tickets zu Aufgaben effizient zu verwalten.
Für die Verwaltung des Projektes wurden zwei solcher Boards erstellt. Auf einem Board wurden Tickets zu organisatorischen Themen verwaltet und auf dem anderen Tickets zu produktiven Themen wie der Entwicklung oder dem Schreiben des Projektberichtes. Das Entwicklungsboard war dabei noch weiter durch Swimlanes, Spalten und Farben unterteilt, um Platz für verschiedene Themen wie Challengr, RedLeaf oder den Bericht zu bieten. In einem sogenannten Backlog war außerdem Platz für Tickets, welche noch keinen Bearbeiter hatten, damit diese keinen Platz auf dem Board einnehmen. So konnten Aufgaben für die Zukunft effizient geplant werden.

[1] https://www.atlassian.com/de/software/jira

Ein Ticket in Jira besitzt einen Titel und eine genauere Beschreibung. Außerdem kann es Mitgliedern des Projektes zugewiesen werden und Komponenten zugeteilt werden, wodurch die zuvor erwähnte Kategorisierung automatisch angewandt wird. Zudem lässt sich an den Tickets die Zeit, welche zur Bearbeitung benötigt wurde, festhalten. Anzumerken ist allerdings, dass den Teilnehmer*innen die Methode zur Zeiterfassung für die Angabe bei den Feedbackgesprächen selbst überlassen war.

2.1.1.2 Git

Git[2] ist ein Versionskontrollsystem, welches es erleichtert, gemeinsam an einer Software zu arbeiten. Es erfasst die Änderungen jeder Person an jeder Datei in einem Repository, sodass Änderungen an derselben Datei zusammengeführt werden oder zu einem vorherigen Stand zurückgekehrt werden kann. Außerdem existiert in Git ein Feature namens Branching, welches das Arbeiten mit verschiedenen Versionen einer Datei gleichzeitig erlaubt. Dieses wurde im Projekt genutzt, um neue Features getrennt vom funktionierenden Softwarestand zu entwickeln und zu testen.
Die Softwares RedLeaf und Challengr, welche im Projekt weiterentwickelt wurden, waren dementsprechend auch Branches der am MEVIS entwickelten Software. Dem Projekt wurde so die Freiheit geboten, nicht nur Features zu entwickeln, welche für alle MEVIS Mitarbeiter von Nutzen sind. Sollten zum Beispiel Features konkret für den Projektnutzen eingebaut werden, so konnten diese beim Merge des Projektbranches auf den MEVIS Branch einfach ignoriert werden. Um keine unvollständigen oder kaputten Features auf den MEVIS Branch zu ziehen, wurde mit Merge Requests gearbeitet. Diese wurden von Projektteilnehmern erstellt und von den Betreuern überprüft und gemerged. Zur Minimierung des Aufwandes am Ende des Projektes wurden fertige Features schon während des Projektes einzeln übertragen.

[2] https://git-scm.com/

2.1.1.3 Entwicklung

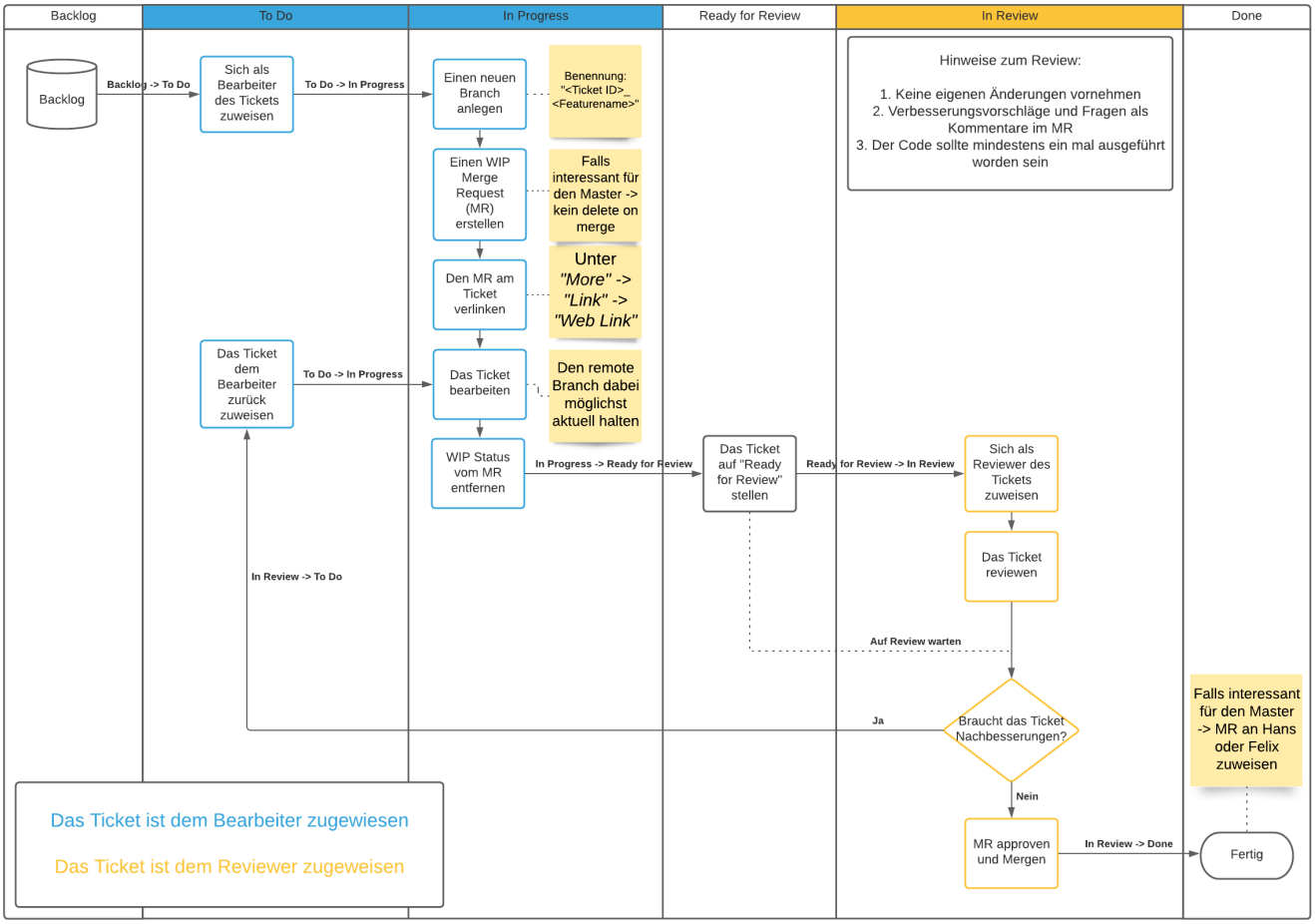


Abbildung 2.1: Git und Jira Workflow

Während der Entwicklung eines neuen Features wurde ein strikter Ablauf vorgegeben, um die Arbeit übersichtlicher und leichter handhabbar zu machen. Dabei wurde je Feature, ein Jira Ticket und ein Branch in Git erstellt, welche stark aneinander gekoppelt waren.
Wird ein Ticket erstellt, so startet es im Backlog und bleibt dort, bis jemand das Ticket reserviert, es in die „To do“-Spalte des Entwicklungsboards schiebt und sich zuweist.
Sobald mit der Arbeit begonnen wird rückt es in die „In Progress“-Spalte. Gleichzeitig wird ein Branch in Git mit zugehörigem Merge Request erstellt. Dieser wird als „Draft“ markiert und an dem Ticket verlinkt. Ist die Arbeit an dem Ticket beendet, so wird der „Draft“ Status von dem Mer-

ge Request entfernt, das Ticket in „Ready for review“ geschoben und der Assignee entfernt.
Nun weist sich jemand das Ticket zu, der dieses reviewen möchte. Das Ticket wird nach „In review“ geschoben und auf mehrere Kriterien überprüft. Die Anforderungen des Tickets müssen erfüllt sein, die Anwendung muss noch fehlerfrei laufen und der Code muss sauber implementiert sein. Sollte es Verbesserungsvorschläge geben, so werden diese am Merge Request vermerkt. Dann wird das Ticket der bearbeitenden Person zurück zugewiesen und in „To do“ geschoben, wo die Vorschläge nach dem bekannten Workflow umgesetzt werden können. Andernfalls gilt das Ticket als erfüllt. In diesem Fall wird der Merge Request gemerged, der Branch gelöscht und das Ticket in „Done“ geschoben.

2.1.2 Plenum

AUTOR*IN: RENE FINZEL

Das Plenum fand einmal wöchentlich in einem Zoom Meeting statt und diente für den offiziellen Austausch zwischen den Student*innen und den Betreuern.
In der ersten Hälfte des Projekts wurde das Plenum dazu genutzt, die Infrastruktur vom MEVIS kennenzulernen, zu lernen, mit MevisLab umzugehen sowie Netze zu erstellen und zu trainieren. In der zweiten Hälfte waren die Themen individueller und es ging hauptsächlich darum den Verlauf des Projekts zu planen und technische Fragen oder Probleme von einzelnen Personen zu klären.
Zu jedem Plenum gab es eine offizielle Plenumsleitung, die alle vier Wochen zwischen den Student*innen wechselte. Somit hatte jede*r einmal die Verantwortung, eine Agenda zu erstellen und während des Plenums durch diese zu leiten.
Dafür wurde jede Woche von der aktuellen Plenumsleitung ein Protokoll in Confluence (siehe Abschnitt 2.1.3.3) angelegt, welches einen Zeitplan, eine Teilnehmerliste und den Ablaufsplan des Plenums beinhalteten. Dieses wurde jede Woche von einer anderen Person geschrieben, welche

zu Beginn des Plenums festgelegt wurde. Fester Bestandteil der Agenda waren die Themen:

- „Organisatorisches“: Hier wurde das weitere Vorgehen des Projekts geplant, Abgabefristen gesetzt oder auch Termine und Pläne für - zum Beispiel - Teambuildingevents besprochen.
- „Weekly“: Im Weekly hatte jede*r die Möglichkeit zu erzählen, was er oder sie die Woche über für das Projekt gemacht hat. Dieser Austausch war sehr hilfreich, um den aktuellen Stand des Projekts einzuschätzen und somit hatte man die Chance darzustellen, an was gerade gearbeitet wurde und ob eventuell Hilfe nötig war.
- „Technische Fragen“: In diesem Abschnitt ging es darum technische Fragen oder Probleme zu Aufgaben zu klären, die zu dem Zeitpunkt bearbeitet wurden.
- „Nach dem Plenum“: Dies war ein inoffizieller Teil des Plenums, an dem die Teilnahme freiwillig war. Hier konnten Fragen gestellt werden, die nicht unbedingt Projektrelevant sein müssen oder nur einzelne Personen betrafen.

Alle Teilnehmer*innen hatte zusätzlich die Möglichkeit, selbstständig Punkte zur Agenda hinzuzufügen oder die bestehenden Themen durch weitere Unterpunkte zu ergänzen.

2.1.3 Kommunikation

AUTOR*IN: TIMO GÜNNEMANN

Im Verlauf des Projektes wurden mehrere Kommunikationsmittel für verschiedene Zwecke verwendet. Für die wöchentlichen Plena wurde Zoom genutzt. Die Universität bzw. die Betreuer haben den Zoom-Raum für die Veranstaltung gestellt. Zoom hat mehrere Funktionen wie z.B. das Freigeben von Bildschirmen und das Aufzeichnen von Meetings. Beide wurden im Verlauf des Projektes häufig genutzt.

2.1.3.1 Mattermost

Um auch außerhalb von den wöchentlichen Plena - sowohl mit unseren Betreuern als auch mit den anderen Projektteilnehmenden - kommunizieren zu können, wurde die Open Source Kollaborationsplattform Mattermost [3] verwendet. Diese ist als Desktop, Browser und App Version verfügbar. Leider war Mattermost aus Datenschutzgründen nach einiger Zeit nur noch über VPN erreichbar. Deswegen wurde für die tägliche Kommunikation Discord [4] anstatt Mattermost verwendet. Mattermost wurde weiterhin genutzt, da es dort mehrere MEVIS interne Kanäle mit allen Mitarbeitenden gibt, wo bei Problemen Fragen gestellt werden konnten. Außerdem werden Logs von laufenden Trainings an Mattermost gesendet, die einen Aufschluss über den Stand des Trainings liefern.

2.1.3.2 Discord

Am Anfang des Projektes haben wir uns einen Discord Server eingerichtet, der nur für die Teilnehmenden des Projektes gedacht war. Nachdem Mattermost nur noch über VPN verfügbar war, haben wir zusätzliche Kanäle eingerichtet, wo auch die Betreuer Zugriff hatten. Somit gab es eine Kategorie für die Kommunikation mit den Betreuern und mehrere Kategorien nur für die Teilnehmenden. Wir haben jeweils eine Kategorie für die einzelnen Themen wie z.B. Challengr, RedLeaf, die KiTS-Challenge usw. erstellt, um den Überblick zu behalten. Das war sehr hilfreich für die Terminabsprachen, Treffen und Kommunikation außerhalb des Plenums sowie Hilfestellung bei Problemen und Ankündigungen von anstehenden Reviews. Außerdem haben wir Rollen für verschiedene Themen wie z.B. die Plenumsleitung verteilt, wodurch man sich einfach an die Rolle wenden konnte und nicht spezifisch an die Person. Das hat auch noch mal zu einer besseren Übersichtlichkeit beigetragen.

[3] https://mattermost.com/
[4] https://discord.com/

2.1.3.3 Confluence

Confluence ist über das MEVIS Intranet zu erreichen und ist eine kollaborative Wissensbasis. Es wurde ein eigener Space für das Projekt Deep Anantomy eingerichtet. Hauptsächlich haben wir Confluence genutzt, um jede Woche die sogenannten Meeting Notes vom Plenum anzulegen, so dass man nachsehen konnte, was in den wöchentlichen Plena besprochen wurde. Außerdem haben wir unsere Experimentenserien von den verschiedenen Deep-Learning-Modellen dort mit den jeweiligen Ergebnissen dokumentiert, um als Ergänzung zu Challengr eine gute Übersicht zu bekommen. Neben den Meeting Notes und Experimentenserien haben wir auch selbst Anleitungen zu Themen wie z.B. Docker, Nutzung des Classifier, Formatieren des Challenger Codes und ein Schaubild zu unserem Workflow angelegt. Im Laufe des Projekts gab es ca. alle drei Monate Evaluationsgespräche, wofür jede*r eine Seite erstellt und anhand einer Vorlage festgehalten hat, woran er/sie im Projekt gearbeitet hat. Des Weiteren wurde Confluence auch genutzt um Ideen und Ziele für das Projekt festzuhalten und um eine Übersicht zu bekommen, wer sich für welche Bereiche interessiert und an der Erreichung dieser Ziele arbeiten möchte.

2.2 Infrastruktur

2.2.1 Übersicht

AUTOR*IN: JANNES ADAM

Am MEVIS steht eine Infrastruktur, bestehend aus Software und leistungsfähiger Hardware, zur Verfügung, die zur Entwicklung von neuronalen Netzen für medizinische Bildverarbeitung dient. Mit MeVisLab können medizinische Bilddaten geladen, manipuliert und visualisiert werden. Dazu sind Schnittstellen zu den anderen Anwendungen vorhanden. RedLeaf ist ein internes Framework für Deep Learning, welches auf den herkömmlichen Deep-Learning-Frameworks basiert. Die Trainings können

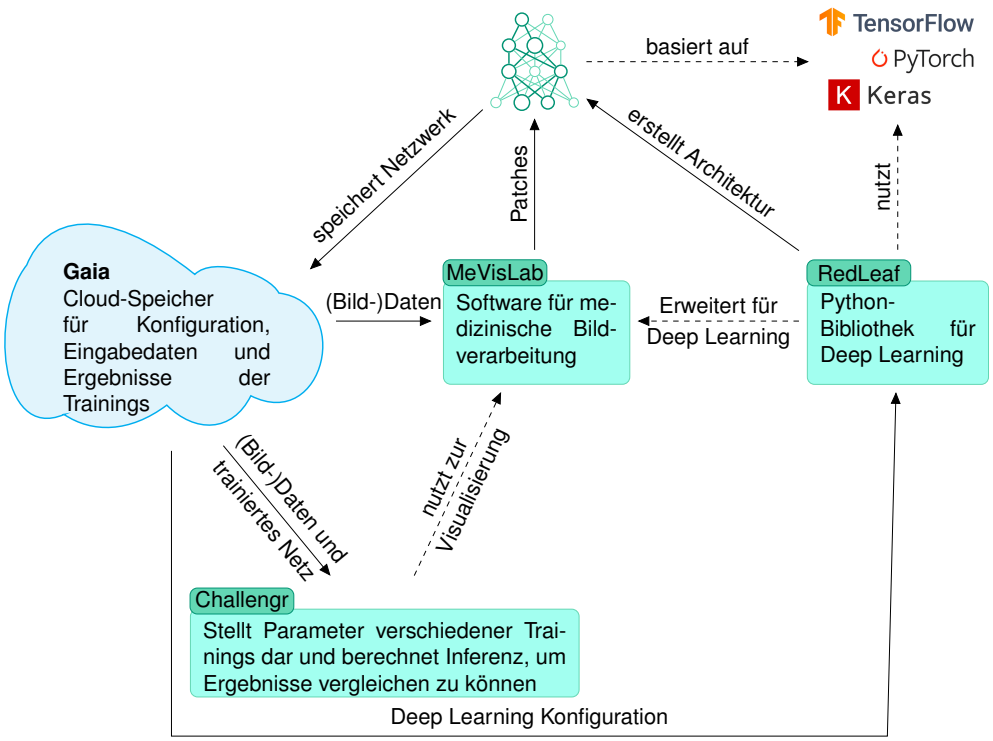


Abbildung 2.2: Übersicht über die Software-Infrastruktur am MEVIS

auf einem Rechner-Cluster gestartet werden, welches auch als Server für Online-Anwendungen dient. Die dazu gehörigen Daten werden auf dem Cloud-Speicher Gaia abgelegt. Zum Vergleich von Trainings kann Challengr genutzt werden, welches die Inferenz berechnet und so einen Vergleich zwischen den Sessions erlaubt. Dazu wird zum Beispiel eine Übersicht über die Parameter bereitgestellt.

2.2.2 MeVisLab

AUTOR*IN: JANNES ADAM

MeVisLab ist eine Software für Bildverarbeitung im medizinischen Bereich, die für Forschung und Entwicklung genutzt wird. Sie ermöglicht schnelles Testen neuer Algorithmen und kann plattformübergreifend für Windows, Linux und MacOS genutzt werden. MeVisLab wird von der MeVis Medical Solutions AG in Zusammenarbeit mit Fraunhofer MEVIS entwickelt. Die Software MeVisLab existiert seit 2004, es gab jedoch schon

ab 1993 erste Entwicklungen. Zur Verwendung existiert eine freie Version für den nicht-kommerziellen Einsatz und eine kommerzielle Lizenz für den uneingeschränkten Einsatz.
MeVisLab ist modular aufgebaut. Es existieren Module für grundlegende Bildverarbeitungsalgorithmen wie z.B. für Segmentierung. Daneben sind spezielle Module für medizinische Bildverarbeitung wie Registrierung vorhanden. So kommen insgesamt mehr als 5000 Module zusammen. Darin enthalten ist auch ein großer Teil des *Insight Segmentation and Registrations Toolkits*, welches eine Sammlung von Algorithmen für Registrierung, Segmentierung und Analyse multidimensionaler Daten ist sowie das *Visualization Toolkit*. Die Software unterstützt das medizintypische DICOM-Format und viele verbreitete Bildformate.
Um in MeVisLab Bilddaten zu verwenden und bearbeiten werden Netzwerke aus Modulen graphisch zusammengesetzt. Durch die modulare Gestaltung ist die Funktionalität leicht erweiterbar. Mit dem *MeVisLab Software Development Kit* können eigene Module entwickelt werden. Diese können mit Python, C++ und Skripten erstellt werden. Dazu enthält MeVisLab eine eigene Entwicklungsumgebung, die u.a. Autovervollständigung, Codehervorhebung und Debugging unterstützt. Module können wiederum ein eigenes Modulnetzwerk beinhalten und so hierarchisch aufgebaut werden. Für Fragen und Probleme existieren Hilfeseiten und ein MeVisLab-Forum.
Die Software basiert auf C++ und verwendet öffentliche Bibliotheken wie QT oder OpenInventor. OpenInventor wird zum Beispiel für die Visualisierung von Daten in 2D und 3D genutzt. Große Volumen, die nicht in den Hauptspeicher passen können mit Hilfe einer speziellen Rendering-Technik visualisiert werden. Für Diagramme wird die Python-Bibliothek matplotlib genutzt [2].

2.2.3 RedLeaf

AUTOR*IN: MARCEL PLUTAT

RedLeaf ist eine von Fraunhofer MEVIS entwickelte Python-Bibliothek für Deep Learning. Der Fokus dieser Bibliothek liegt auf Remote Deep Lear-

ning, was unter anderem der Name bereits unterstreicht, da RedLeaf für „**Re**mote **D**eep **Lea**rning **F**ramework“ steht. Remote bezieht sich dabei nicht auf andere Clients im selben Netzwerk, sondern auf die Verbindung zu MeVisLab. RedLeaf ist Remote, da Teile in MeVisLab oder anderweitig laufen. Aus diesem Aspekt wird ebenfalls bereits deutlich, dass RedLeaf für den Einsatz in Verbindung mit MeVisLab entworfen wurde.
RedLeaf wurde in Python implementiert und verwendet Remote Procedure Calls (RPC) für die Verbindungen zu MeVisLab. RedLeaf selbst beinhaltet MeVisLab-Module, welche zum Beispiel für die Bildsegmentierung in ein MeVisLab-Netzwerk eingebunden werden können. Diese Module befassen sich beispielsweise mit den Remote-Verbindungen wie der *Remote Patch Server* oder mit dem Ausschneiden von Bildsegmenten (Patches) aus den medizinischen Bilddaten durch das *Extract Patches*-Modul.
Außerdem sind in RedLeaf Deep Learning Architekturen implementiert. Diese basieren auf den Methoden der drei großen Python Deep Learning Frameworks, *Tensorflow*, *Keras* und *PyTorch*. Beispiele für Deep Learning Architekturen sind U-Nets oder von uns implementierte Netzarchitekturen, die in Kapitel 4.2 beschrieben sind.
Weitere Komponenten von RedLeaf sind Pre- und Postprocessing-Methoden. Im Preprocessing kann beispielsweise Data Augmentation als MeVisLab-Modul oder direkt in RedLeaf verwendet werden. Weiterer Teil des Preprocessings sind die sogenannten „Tasks“, welche den Gesamtablauf des Trainings durch eine festgelegte Aufgabe steuern. Die „Tasks“ definieren beispielsweise eine Segmentierung im *SegmentationTask* oder eine Klassifikation durch den *ClassificationTask*. Ein typisches Postprocessing-Modul ist die Umwandlung von One-hot kodierten Vektoren wie der Netzausgabe in Integer Werte. Der letzte große Teil von RedLeaf sind Methoden, um das Training zu überwachen. RedLeaf loggt dafür die Metriken des Trainings wie Loss-Werte, Akkuratheit und Jaccard-Scores, etc., durch welche ein Plot des Trainingsverlaufs erstellt werden kann.

Abbildung 2.3 gibt einen Überblick der RedLeaf-Komponenten. Grundsätzlich ist darauf zu erkennen, dass sich die Komponenten in zwei Bereiche aufteilen lassen. Erstens beinhaltet RedLeaf das Training von Netzen und zweitens kann über einen Klassifikationsserver die Inferenz von

Netzen erfolgen. In beiden Fällen ist zu erkennen, dass RedLeaf mit MeVisLab verbunden ist, zum Beispiel um einen Batch an Trainingsdaten zu erhalten. Der weitere Ablauf ist in beiden Fällen ähnlich. Zuerst erfolgt das Preprocessing, dann das Training respektive die Inferenz eines Netzes und letztlich folgt das Postprocessing. Diese drei Schritte decken den Klassifikationsserver ab. Das Training beinhaltet allerdings noch einen zusätzlichen Aspekt, nämlich das Logging von Daten und die Überwachung des Trainings durch ein Monitoring Tool. In der Praxis ist dieses Monitoring zum Beispiel der Live Loss Plot auf dem GPU-Cluster.

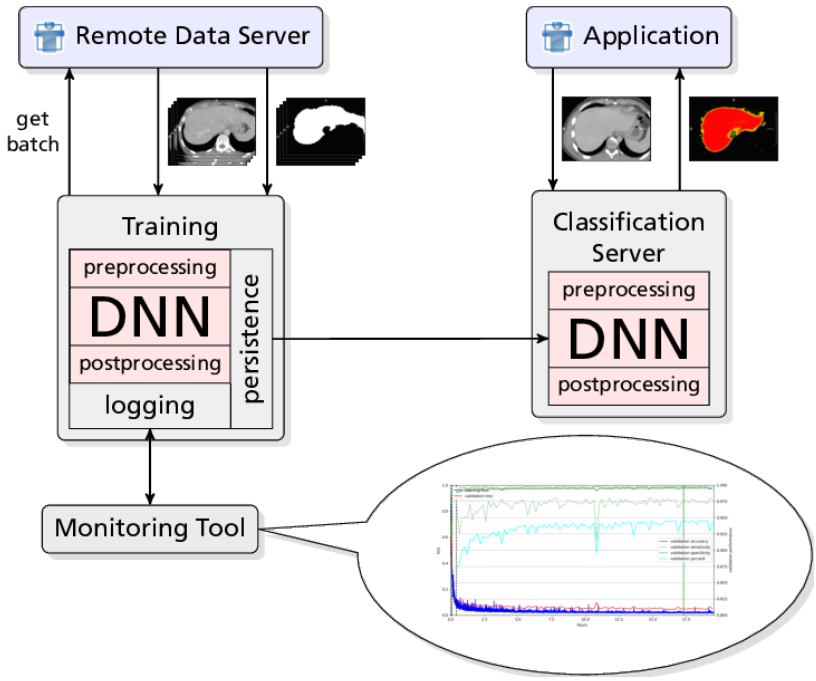


Abbildung 2.3: RedLeaf Komponenten aus [3]

2.2.4 Docker

AUTOR*IN: ROBERT BOHNSACK

Um das Bereitstellen und Nutzen von eigener Software am MEVIS zu vereinfachen, wird Docker verwendet. Docker ermöglicht es, Software in Containern zu isolieren, welche alle gewünschten Abhängigkeiten bereits enthalten.

Am MEVIS wird dies speziell für Images von RedLeaf, Challengr und MeVisLab genutzt. Diese Images können lokal verwendet werden, dienen jedoch hauptsächlich dazu, die Nutzung der Software auf dem Cluster (siehe Kapitel 2.2.5) zu ermöglichen.
Wird ein Image in einer Konfiguration, welche auf dem Cluster laufen soll, anhand von Name und Tag angegeben, so wird dieses bei der Ausführung aus der lokalen Registry geladen und ausgeführt. So kann zum Beispiel ein RedLeaf-Image für Trainings oder ein Challengr-Image für ein immer verfügbares Challengr-Frontend verwendet werden.
Um auch dem Projekt das Bereitstellen eigener Images zu ermöglichen, wurde in der Registry ein Bereich eingerichtet, auf dem die Mitglieder Schreibrechte hatten. Dieser wurde weiter in zwei Bereiche unterteilt: zum einen Images für Challengr und RedLeaf, welche im Normalfall genutzt wurden, zum anderen Test-Images für Challengr und Redleaf, welche meist dazu dienten, neuen Code auf dem Cluster zu testen.

2.2.5 Cluster

AUTOR*IN: MARKUS RINK

Das Rechnercluster verfügt über neun Rechner mit jeweils vier GTX 1080 Ti Grafikkarten, welche über Nomad verwaltet werden. Nomad ermöglicht den Zugriff auf die verschiedenen Rechner über eine einheitliche Schnittstelle und automatisiert die Ressourcenverteilung. So können Docker-Container gestartet werden, die Neuronale Netze trainieren, Inferenzen ausführen oder Web-Apps hosten.

Diese Schnittstelle wird auch von dem QuantMed-Dashboard genutzt, welches automatisch eine Konfigurationsdatei für ein Training auf dem Cluster generiert und es startet. Über das Dashboard wird laufend der aktuelle Stand des Trainings angezeigt und es ermöglicht einen direkten Zugriff in den im Docker-Container liegende MeVisLab-Prozess.

2.2.6 Gaia

AUTOR*IN: MARKUS RINK

Gaia bezeichnet den Datenserver des Instituts. Dieser Netzwerkspeicher ist von allen Projektteilnehmenden und dem Cluster erreichbar. Der Input-Ordner speichert die Netzkonfigurationen und Trainingsdaten. Der Output-Ordner speichert die trainierten Netzgewichte. Als „good practice“ haben wir die Trainingsordner mit Git versioniert und den Master-Branch per Pipeline mit Gaia synchronisiert, sodass Änderungen rückgängig gemacht werden können. Diese Funktion wurde für MeVisLab-Inferenz- und Evaluationsnetzwerke genutzt, aber nur zum Teil für Architekturen.

2.2.7 Challengr

AUTOR*IN: TIMO GÜNNEMANN

Challengr ist ein MEVIS internes Tool, welches dazu dient, trainierte Modelle miteinander zu vergleichen. Es basiert auf dem Client-Server Modell. Das Backend ist in Python programmiert und besitzt eine REST-Schnittstelle zum Frontend. Das Frontend ist in Javascript programmiert mit Quasar [5] als Framework, welches auf Vue.js basiert. Um Modelle miteinander vergleichen zu können, wird eine Challenge erstellt, der die Modelle zugeordnet sind. Die Challenge kann an die jeweiligen Bedürfnisse angepasst werden, was die Test- und Referenzdaten und die Evaluationsmaße umfasst.

Um Modelle miteinander vergleichen zu können, wird zuerst eine Inferenz auf dem Testdatensatz durchgeführt und die Ergebnisse dann gegen die Referenzdaten evaluiert. Das Inferenznetzwerk und die zu berechnenden Metriken werden per MeVisLab-Netzwerk definiert. Die Ergebnisse werden dann im Webinterface dargestellt. Hier können aus der Liste aller Modelle in dieser Challenge genau zwei Modelle gewählt werden, um diese miteinander zu vergleichen. Für den Vergleich dient eine große Tabelle,

[5] https://quasar.dev/

wo alle Parameter und die Evaluationsmaße gegenübergestellt werden. In einem Scatter-Plot werden die Ergebnisse der einzelnen Fälle visualisiert. Die Case-Data-Table dient dazu, jeden einzelnen Fall der gewählten Modelle anhand der Evaluationsmaße miteinander zu vergleichen. Außerdem können die Ergebnisse pro Fall mit der CaseVisualization in MeVisLab visualisiert werden.

3 Theoretischer Hintergrund

3.1 Grundlagen Neuronale Netze

AUTOR*IN: LENA PHILIPP

3.1.1 Semantische Segmentierung

Im Gegensatz zur Klassifizierung von Bildern wird bei der Segmentierung nicht ein bestimmtes Label für das jeweilige Eingabebild ausgegeben, sondern eine Klasse pro Pixel. Computergestützte Segmentierung von Bilddaten spielt in der medizinischen Bildverarbeitung eine zentrale Rolle bei der Unterscheidung verschiedener anatomischer Strukturen. Außerdem können medizinische Bildobjekte vermessen und Gewebeveränderungen erkannt werden. Die Art der Problemstellung, beziehungsweise der Ausgabe des Netzes, beeinflusst die Netzarchitektur.

Nach der Segmentierung einzelner zusammenhängende Bereiche können diese klassifiziert werden, beispielsweise zur Trennung von pathologischem und gesundem Gewebe.

3.1.2 Neuronale Netze

Deep Learning ist ein Teilgebiet des maschinellen Lernens. Ziel ist es Zusammenhänge aus Daten zu lernen. Dabei werden **neuronale Netze** verwendet. Dies sind parametrisierte Modelle, die aus mehreren, hierarchisch aufgebauten Schichten bestehen [4]. Die Schichten setzen sich zusammen aus kleinen Bausteinen, den Neuronen. Diese bekommen

Eingabewerte, die gewichtet werden und einen Bias. Eine Aktivierungsfunktion bestimmt daraus die Ausgabe. Die Gewichte und der Bias sind die lernbaren Parameter des Netzes, die zunächst zufällig initialisiert werden. Als **Tiefe** eines Netzes wird die Anzahl der Schichten bezeichnet. Die erste Schicht ist die Eingabe der Daten, die dann durch die folgenden (versteckten) Schichten weiterverarbeitet werden, bis zur letzten Schicht, der Ausgabe. Die Neuronen einer Schicht werden mit einheitlicher Aktivierung zusammengefasst.
Diese muss differenzierbar sein. Eine gebräuchliche **Aktivierungsfunktion** ist zum Beispiel Rectified Linear Unit (ReLU) oder Sigmoid. Lange Zeit war der Gebrauch von der logistischen Funktion Sigmoid der „Quasistandard“. Lineare Aktivierungsfunktionen geben den Wert proportional zur Eingabe weiter, während die Sigmoid-Funktion die Eingabewerte nicht-linear transformiert auf Werte zwischen 0 und 1. **ReLU** ist der linearen Funktion ähnlicher. Der wesentliche Unterschied liegt darin, dass 0 zurückgegeben wird für negative Werte und Werte gleich 0 [5] .

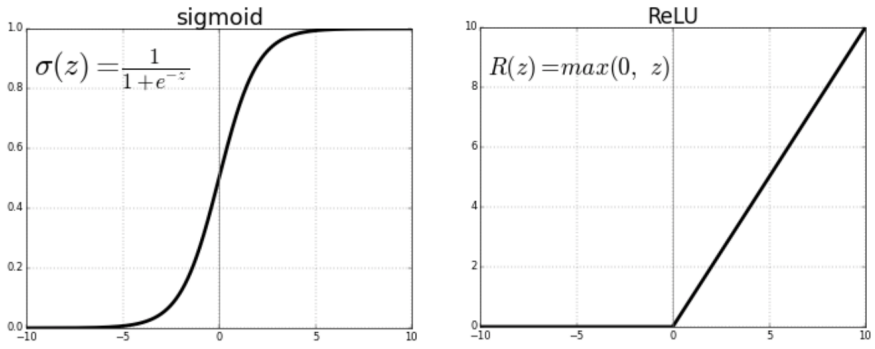


Abbildung 3.1: Aktivierungsfunktionen ReLU und Sigmoid [6]

3.1.3 Convolutional Neural Network

Eine Operation, die besonders in der Bildverarbeitung eine wesentliche Rolle spielt, ist die Faltung. Durch das Approximieren von Gradienten können Strukturen wie Ecken erkannt werden. Durch das Nutzen von Faltungen können neuronale Netzen außerdem erkennen, wie Strukturen räumlich zusammenhängen.

Allgemein bezeichnet eine Faltung die Verbindung von zwei Funktionen. Bei der Faltung bei neuronalen Netzen handelt es sich bei um eine Matrixmultiplikation der Eingabe und einem **Filter**, zu der ein Bias hinzuaddiert wird. Das Ergebnis dieser Operation wird oft als **Feature Map** bezeichnet. Typische Maße von Filtern sind 3×3 oder 5×5 und damit meist wesentlich kleiner als das Eingabebild. Dies ist die Größe des **rezeptiven Feldes** eines Neurons, das dem biologischen rezeptiven Feldes dahingehend ähnelt und wie dieses ebenfalls auf einen lokal eingegrenzten Bereich reagiert. Ein Beispiel zeigt Abbildung 3.2. Durch die Schichtung kommt es durch eine Vergrößerung des Feldes wie Abbildung 3.3 verdeutlicht. Neben der Filtergröße bestimmt auch die Tiefe des Netzes wie groß der Bereich der Eingabe ist, der zur Berechnung eines Pixels in der Ausgabe hinzugezogen wird [7].

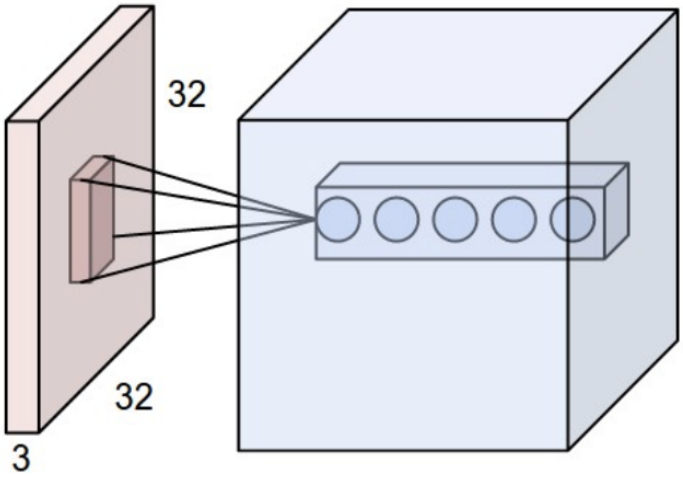


Abbildung 3.2: Links wird ein $32 \times 32 \times 3$ Eingabebild dargestellt und rechts Neuronen einer Convolutional Schicht. Jedes Neuron ist räumlich nur mit einem Teil des Bildes, einem lokalen Bereich, verbunden - aber mit der vollen Tiefe. Diese fünf Neuronen besitzen eigene Gewichte, beziehen sich aber auf das gleiche rezeptive Feld [8]

Für die Berechnung der Feature Map wandert der Filter entlang der Breite und Höhe der Eingabe mit einer bestimmten **Schrittweite** (Stride) über die Eingabematrix. Die Werte beider Matrizen werden nun multipliziert und das Ergebnis in der zwei-dimensionalen Ausgabe gespeichert.

Die Schrittweite beeinflusst wie sehr sich die Ausgabe gegenüber der Eingabe verkleinert. Um die räumliche Größe der Eingabe in der Aus-

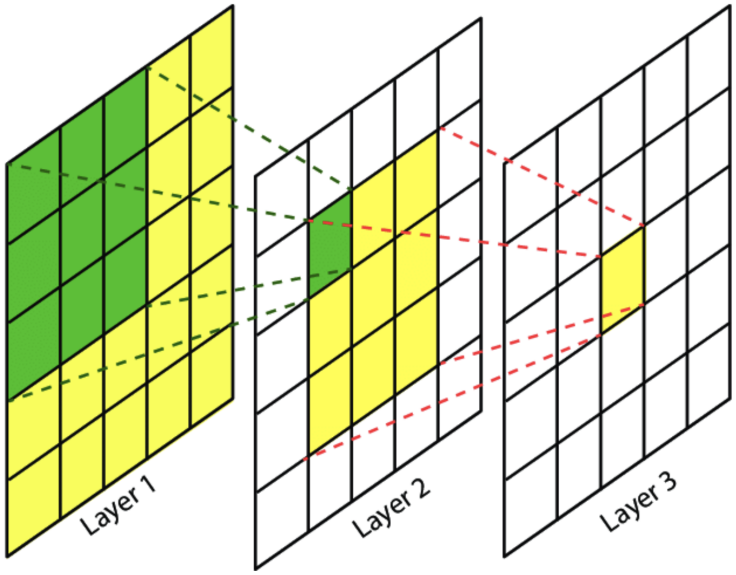


Abbildung 3.3: Dargestellt werden drei Filter mit der Größe 3×3. Während die grüne Fläche das rezeptive Feld eines Pixels aus der zweiten Schicht umfasst, zeigt die gelbe Fläche das rezeptive Feld eines Pixels der dritten Schicht über die Schichten hinweg [7]

gabe beizubehalten werden am Rand Werte hinzugefügt, beispielsweise Nullen - das **Zero Padding**. Die Werte des Filters bleiben gleich bei diesem Vorgang, sodass Merkmale extrahiert werden, die daher translationsinvariant sind. Die Werte für diese Filter werden aus den Daten gelernt. Meist werden innerhalb einer Faltungsschicht mehrere Filter verwendet, die auf mehrere Merkmale reagieren. Dadurch werden auch mehrere Feature Maps angelegt, die dann entlang einer Dimension zusammengefasst werden. So entsteht ein Ausgabetensor [10].
Ein wichtiges Konzept der Faltungsschicht ist das des Parameter Sharings. Die Neuronen einer Feature Map teilen sich die gleichen Gewichte, sodass zum Einen die Zahl der lernbaren Parameter kontrolliert wird und zum Anderen wird so für bestimmte Bereiche überprüft ob bestimmte Merkmale oder Konzepte vorhanden sind [11].

3.1.4 Pooling

In einem Convolutional Neural Network folgt einer oder mehreren Faltungsschicht(en) in der Regeln eine Pooling-Schicht. In dieser werden be-

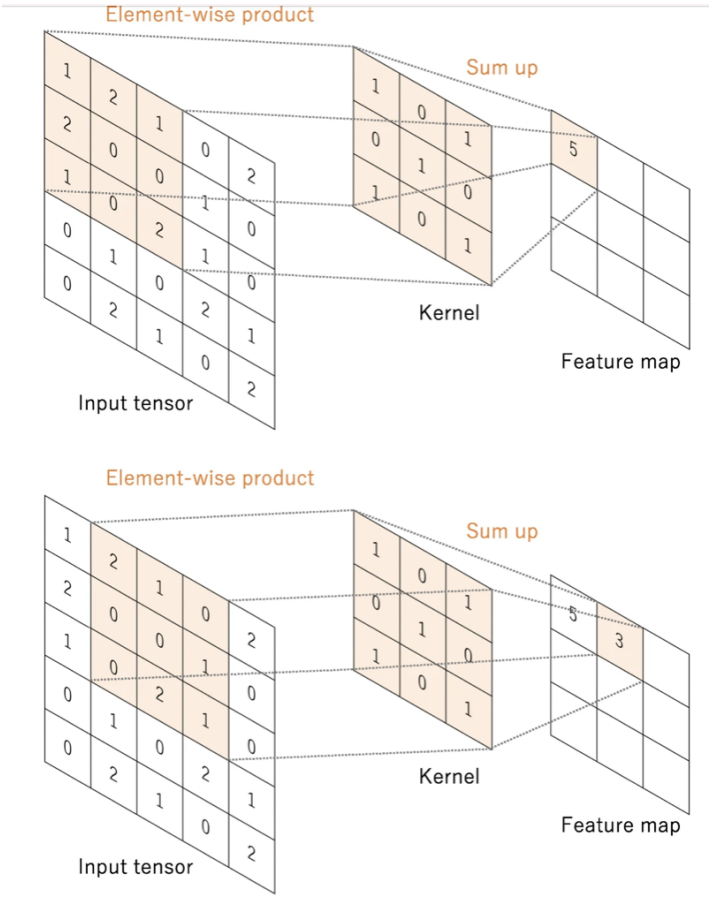


Abbildung 3.4: Ein Beispiel für eine Faltungsoperation, bei dem das Eingabebild mit dem 3×3 Filter / Kernel jeweils Element für Element multipliziert wird. Die Ergebnisse dieses Schrittes werden im Ausgabetensor / Feature Map abgelegt und summiert. Es wird kein Padding verwendet und ein Stride von 1, wie im unteren Bild zu sehen ist [9].

nachbarte Merkmale aus dem Tensor zusammengefasst. Dadurch wird zum einen der Speicher- und Rechenbedarf gesenkt und durch das Zusammenfassen wird die Representation zum Einen auf Wesentliches reduziert und dazu annährend **translationsinvariant**. Die genaue Position des Merkmals ist weniger wichtig als der Umstand, dass ein Merkmal vorhanden ist [12].

Darüber hinaus wird das rezeptive Feld über die Schichten hinweg vergrößert. Es gibt verschiedene Arten das Pooling durchzuführen, zum Beispiel indem das Maximum der Werte aus einem festgelegten Bereich bestimmt wird (**Max Pooling**, Abbildung 3.5) oder der Durchschnitt. Auch hier kann die Schrittweite gewählt werden, die Einfluss darauf hat, wie

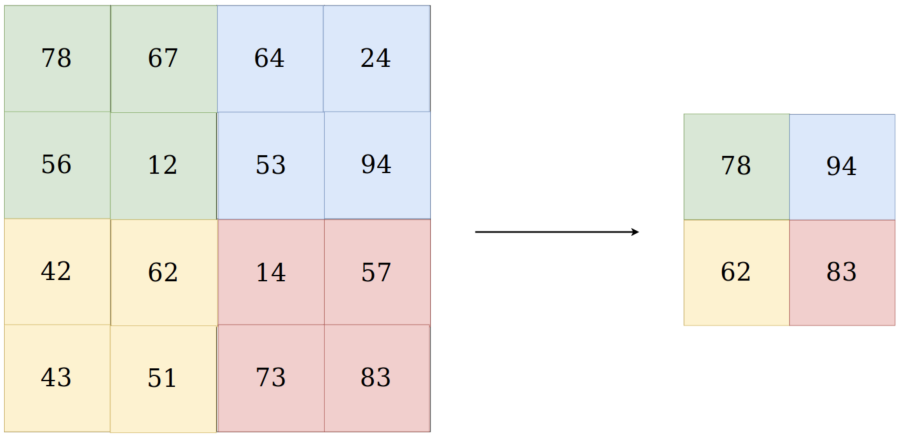


Abbildung 3.5: Max Pooling [13]

groß die Verringerung von Ein- zu Ausgabe der jeweiligen Schicht ist [14]. Neben den bereits erwähnten Arten Pooling zu verwenden spielt das Global Average Pooling eine besondere Rolle. Dabei wird der Durchschnitt aller Elemente einer Feature Map für Höhe x Breite bestimmt und so auf eine Größe von 1×1 verkleinert. Die Tiefe dieser bleibt dabei erhalten. Dadurch wird unter anderem die Anzahl der lernbaren Parametern verringert und ermöglicht dem Netz mit Eingaben verschiedener Größen zu arbeiten. Global Average Pooling wird in der Regel vor der Fully Connected Layer angewendet [13].

3.1.5 Fully Connected Layer

Die Fully Connected Layer wird auch Dense Layer genannt. In dieser Art von Schicht sind alle Neuronen zwischen den Schichten miteinander verbunden, die Neuronen innerhalb einer Schichten jedoch nicht. Fully Connected Layers sind die Basis für Fully Connected Networks. Diese finden Anwendung bei Klassifikationsaufgaben. Für die Eingabe wird dann zum Beispiel eine Wahrscheinlichkeit für eine bestimmte Klasse ausgegeben [15].

Bei vielen Convolutional Neural Networks wird das Eingabebild durch die Abfolge von verschiedenen Convolution und Pooling Schichten verkleinert, während die Anzahl der Filter erhöht wird. Am Ende stehen eine

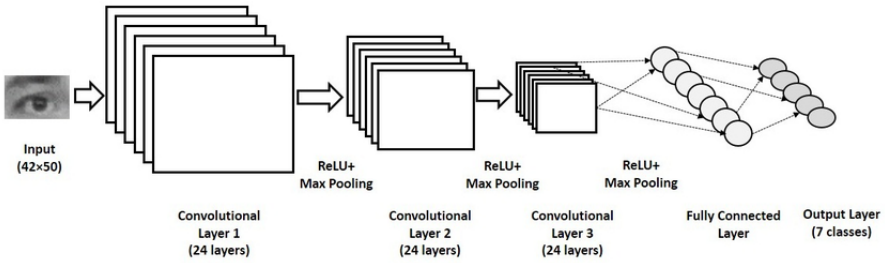


Abbildung 3.6: Beispiel für die Struktur eines Convolutional Neural Networks einschließlich Flattening und Fully Connected Layers [16]

oder mehrere Fully Connected Schichten. Damit das Ergebnis der vorhergehenden Schichten durch diese verbundene Schicht oder Schichten weiterverarbeitet werden kann, muss die Ausgabe ausgerollt werden. Dabei wird das Ergebnis in einen eindimensionalen Vektor transformiert, was man **Flattening** nennt. Ein Beispiel dafür zeigt Abbildung 3.6. Wie bereits unter 3.1.5 kann vor der Fully Connected Layer auch Global Average Pooling anstelle von Flattening verwendet werden.
Die letzte Schicht ist abhängig von der Aufgabe des Netzes. Für Klassifikation werden an dieser Stelle zum Beispiel so viele Neuronen wie zu bestimmende Klassen verwendet. Als Aktivierungsfunktion kann **Softmax** angewendet werden für ein Multi-Klassenproblem, dann würden sich die Ausgabe aller Neuronen zu 1 addieren. Durch Softmax werden die Ausgaben normalisiert und können als Wahrscheinlichkeiten für die Klassen interpretiert werden [17] [13].

3.1.6 Trainingsprozess

Als Lernen wird der Prozess des iterativen Findens von zu der Fragestellung passenden Parameter verstanden. Es gibt verschiedene Lernverfahren. Zwei große Kategorien zur Einteilung verschiedener Ansätze sind dabei überwachte / diskriminative und unüberwachte/ generative Verfahren. Beim **überwachten Lernen** sind Zielvariablen vorhanden, die durch die Eingabevariable vorhergesagt werden soll. Dies passiert indem das Netz eine Eingabe erhält, in der Regel ein Batch bestehend aus mehreren Datenpunkten, und die produzierte Ausgabe durch eine Fehlerfunktion bewertet wird. Als **Fehlerfunktionen** kann zum Beispiel

Mean Squared Error verwendet werden oder die für semantische Segmentierung typischen Fehlerfunktionen Cross Entropy, Focal Loss oder Dice Loss [18]. Beim **Dice Loss** wird der Dice Koeffizient verwendet und von 1 subtrahiert. Der Dice Koeffizient misst die Ähnlichkeit von zwei Bildern und gibt Werte zwischen 0 und 1 aus. Je höher dieser Wert, desto größer die Übereinstimmung. Subtrahiert man diesen Wert von 1, wird die Differenz der Bilder minimiert [19]. Der Fehler, die durch die Fehlerfunktion ausgegeben wird, wird im Training minimiert durch Gewichtsanpassung mit dem Gradientenabstiegsverfahren **Backpropagation**. Dabei wird rückwärts der Anteil der Neuronen in den versteckten Schichten zum Fehlerwert berechnet und entgegen dem Fehler angepasst [20].
Bei **Class Weighting** wird der Fehler der Lossfunktion gewichtet. Damit können einzelne schlechte Labelklassen niedriger gewichtet werden oder ganze Bereiche gänzlich ignoriert werden. Entweder wird der Fehler mit einer Maske in größe der Ausgabengröße des Netzes multipliziert oder die Klassen werden in der Lossfunktion unterschiedlich gewichtet [21].

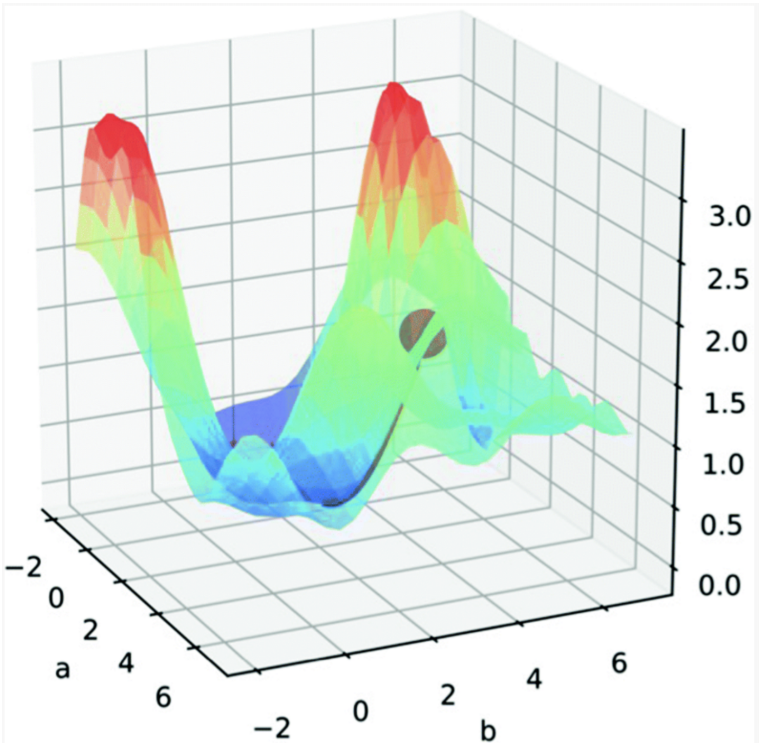


Abbildung 3.7: Die Lernrate bestimmt wie groß die Schritte sind, die auf der Fläche vorgenommen werden, um den tiefsten Punkt zu erreichen. Der tiefste Punkt steht für das Minimum, dass beim Gradientenabstieg gefunden werden soll [5].

Dieser Prozess wird durch die **Lernrate** beeinflusst. Durch diese wird festgelegt wie groß die Schritte sind, die bei der Anpassung vorgenommen werden, wie Abbildung 3.7. Die Rate wirkt sich außerdem auf die Dauer des Trainings aus. Vor Beginn des Trainings wird der Trainingsdatensatz in kleinere Segmente aufgeteilt und die Größe dieser Batches beeinflusst ebenfalls die Trainingsdauer. Während höhere **Batchgrößen** dazu führen, dass der Gradient genauer abgebildet werden kann, fügen kleine Größen unter Umständen Noise zum Lernprozess hinzu. Dies kann sich positiv auf die Übertragungsleistung des Netzes auswirken. Für ein Training mit niedriger Batchsize ist aus Stabilitätsgründen eine kleine Lernrate ratsam. Die Wahl der maximalen Batchgröße hängt von der verfügbaren Rechenleistung ab. Der vollständige Durchlauf aller Daten wird als **Epoche** bezeichnet und die Anzahl der **Iterationen** gibt an wie viele die Batches das Netz durchlaufen haben [5] [22].
Die Lernrate wird entweder fix gewählt oder durch **Optimizer** angepasst. Ein solcher Optimizer ist zum Beispiel Adam (Adaptive Moment Estimation). Für verschiedene Parameter werden individuelle und adaptive Lernraten berechnet. Diese basieren auf Schätzungen von Momenten niedriger Ordnung [23]. Es werden zwei Methoden kombiniert, Stochastic Gradient Descent mit Momentum und Root Mean Square Propagation (RMSprop). Damit findet eine Skalierung der Lernrate statt durch exponentielle Gewichtung des Durchschnitts und es wird der moving average verwendet. So wird mit Adam die Schrittgröße beim Gradientenabstieg adaptiv, sodass ausreichend große Schritte dazu führen, dass lokale Minima überwunden werden können und minimale Oszillationen entstehen beim Erreichen von globalen Minima [24] [25].

3.1.7 Hyperparameter

Für das Training eines neuronalen Netzes müssen viele Entscheidungen getroffen werden, die von der Aufgabenstellung und den Daten abhängig sind. Unter Hyperparametern werden die Parameter verstanden, die vor Beginn des Trainings gesetzt werden und den Prozess des Lernens beeinflussen. Diese werden nicht aus dem Datensatz gelernt. Es gibt Hyperparameter, die die Optimierung des Gradientenabstiegs betreffen

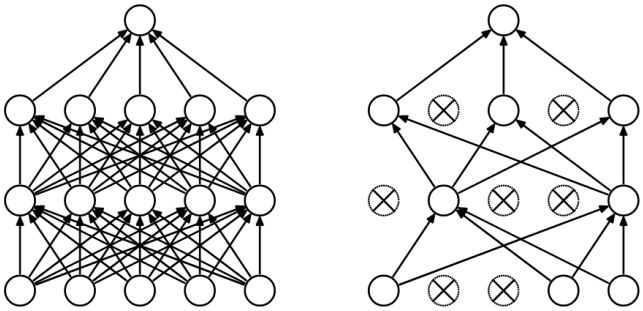
und welche, die das Modell an sich betreffen. Zu der ersten Gruppen gehört zum Beispiel die Lernrate und die Batchgröße sowie die Anzahl an Iterationen. Das Modell betreffend sind unter anderem relevante Hyperparameter die Tiefe, die Aktivierungs- und Fehlerfunktion sowie Art der Schichten und Maßnahmen zur Regularisierung [5] [22].

3.1.8 Regularisierung

Ein zentrales Problem von neuronalen Netzen ist der Tradeoff zwischen Bias und Varianz. Je komplexer das Modell, desto besser können die Daten abgebildet werden. Jedoch kann es bei zu hoher Varianz des Modells passieren, dass sich das Modell zu sehr auf die Trainingsdaten anpasst und keine echten Zusammenhänge lernt. So lernt das Netz das Rauschen der Trainingsdaten, das außerhalb dieser nicht existiert. Hierbei spricht man von Overfitting. Die gelernten Muster lassen sich schlecht auf unbekannte Daten generalisieren. Das gegenteilige Underfitting tritt ein, wenn das Modell einen hohen Bias hat und die Muster innerhalb der Trainingsdaten unzureichend repräsentiert [5]. Durch das Verbinden mehrerer Schichten von Neuronen über Gewichte besitzen Neuronale Netze eine Vielzahl von freien Parametern und häufig ist die Menge an Trainingsdaten nicht ausreichend um Overfitting zu verhindern. Um die Generalisierungsleitung eines Netzes zu verbessern können Regularisierungsstrategien angewendet werden. Bei vielen Ansätzen geht es darum die Kapazität des Netzes zu verringern. Hierfür wird beispielsweise Batch Normalization, Dropout oder Data Augmentation verwendet.

Wenn **Dropout** in ein Netz integriert wird, dann werden verschiedene ausgedünnte Variationen des gleichen Netzes trainiert und miteinander kombiniert. Dies geschieht durch das temporäre Entfernen von Neuronen während des Trainings samt der ein- und ausgehenden Verbindungen. Welche Neuronen ausfallen ist zufällig und an eine festgelegte Wahrscheinlichkeit geknüpft, die Dropout Rate. Der Trainingsprozess verlangsamt sich dadurch, da nicht immer alle Gewichte angepasst werden können bei der Backpropagation. Beim daraus entstehenden Modell stehen schließlich alle Neuronen zur Verfügung und die Wahrscheinlichkeiten

werden mit den Gewichten multipliziert. Overfitting soll so auf zwei Arten verhindert werden: durch die Kombination der Vorhersagen verschiedener Netze und durch das Verhindern von Co-Adaption von Merkmalen [26].



(a) Standard Neural Net (b) After applying dropout.

Abbildung 3.8: Links: Netz ohne Droput. Rechts: Netz mit Dropout [26].

Durch **Data Augmentation** soll dem Netz mehr Daten zur Verfügung gestellt werden. Die Idee, auf der dieser Ansatz basiert, ist, dass die Generalisierungsleistung eines Modells verbessert wird, wenn mehr Daten verwendet werden können um dem Parameterüberschuss beim Training entgegen zu wirken [27]. Der Trainingsdatensatz wird durch verschiedene Operationen, wie geometrische Transformationen, das Hinzufügen von Rauschen, zufälliges Entfernen von Pixeln oder über das Nutzen von Generative Adversarial Networks (GAN) erweitert [28]. Durch GANs werden Bilder generiert und dies kann dazu beitragen einen Ausgleich bei unbalancierte Klassenverhältnisse zu finden [29].

In Neuronalen Netzen benutzt man häufig Normalisierungsebenen in Form von **Batch Norm** um den Vorteil von Normalisierung auch innerhalb des Netzes zu nutzen. Batch Normalisierung wird beispielsweise an Convolutional Schichten angeschlossen. Beim Training werden zunächst die Eingabedaten normiert durch den Mittelwert und die Standardabweichung, die sich aus dem jeweiligen Batch ergeben. Das Ergebnis wird mit einem Skalierungsfaktor multipliziert und einer Konstanten addiert, die im Training gelernt werden [15]. Ioffe und Szegedy stellten 2015 diese Methode vor und zeigen dabei, dass das Training beschleunigt wird [30]. Während des Trainings verändert sich die Verteilung der Eingabe jeder

Schicht durch die Veränderung der Parameter der vorherigen Schicht. Dies benennen die Autoren als interne Kovariatenverschiebung, wodurch das das Training erschwert wird. Die Normalisierung der Eingabe wirkt dem entgegen. Shibani Santurkar et al. [31] argumentieren, dass Batch Norm die Loss Oberfläche glättet und so einen stabileren und effektiveren Lerngradienten erzeugt. Die Erklärung warum sich durch Verwendung von Batch Norm Verbesserungen zeigen, ist weiterhin untersucht. Außerdem können regularisierende Effekte beobachtet werden, die damit erklärt werden können, dass bei der Normalisierung Rauschen hinzugefügt wird [12].

Eine weitere gängige Methode der Regularisierung ist das **Early Stopping**. Dabei wird als Grundlage das Trennen der Daten in Training und Validierung vorgenommen. Das Training wird gestoppt (bzw. der Zeitpunkt bestimmt), sobald der Fehler auf den Validierungsdaten steigt. Dadurch soll der Generalisierungsfehler minimal gehalten werden. Der Stand der Gewichte des Netzes von dem vorhergehenden Zeitpunkt im Training wird als bestes Modell festgehalten [32].

3.2 Preprocessing

AUTOR*IN: MARKUS RINK

CT Bilder sind 3D Volumen aus Grauwerten, gemessen in **Hounsfield Units (HUs)**. Sie liegen im Wertebereich von $[-1000, +1000]$, wobei manche Scanner einen größeren Bereich ausgeben. Die 2D Bilder bzw. Slices, in denen man CTs betrachtet, können in transversaler, coronaler und sagittaler Ebene ausgeschnitten sein (vgl. Abbildung 3.9). Für die Bildverarbeitung hat die transversale Ebene den Vorteil, dass außerhalb des Körpers nur Luft erwartet wird und fehlende Voxel mit -1000HU gefüllt werden können. Dies gilt für die anderen Ansichten nur, falls ein Ganzkörperscan vorhanden ist. Auch andere Strukturen im Körper haben feste Werte im HU Wertebereich (vgl. Tabelle 3.1). Beim Einsatz von Kontrastmittel ändert sich diese Zuordnung, weswegen man bei der Bildverar-

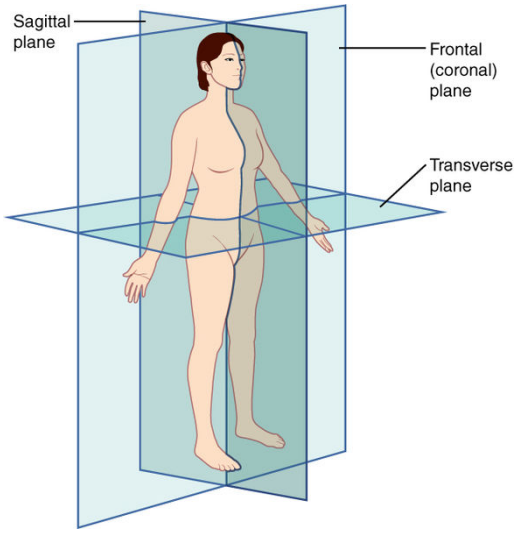


Abbildung 3.9: Körperebenen
OpenStax College. Anatomy & Physiology [33]

beitung darauf achten muss, ob der Datensatz Fälle mit Kontrastmittel enthält.

In der Computertomographie wird die Samplingrate, also die Auflösung des Scans, je nach Fall gewählt, wodurch sich die Längen der Voxel pro Aufnahme unterscheiden. Auch sind die Längen meist in den Dimensionen unterschiedlich. Durch ein **Resampling** auf einen einheitliche Voxelgröße ist das Rezeptive Feld für alle Fälle gleich groß, was das Training erleichtert. Je strukturierter der Datensatz ist, desto weniger muss das Netzwerk beim Training lernen. Je nach Resamplingstrategie können Bildartefakte in die Daten gelangen. Für Label wurde Trilineare und

Hounsfield Units	**Struktur**
>1000	Knochen
50 bis 70	Nieren
50 bis 60	Leber
20 bis 40	Fettleber
10 bis 20	Zysten
0	Wasser
-100 bis -200	Fett
-600 bis -800	Lungengewebe
< -1000	Luft

Tabelle 3.1: Elemente in der HU Skala
Grundlagen der Diagnostik und Intervention [34]

für Bilder Lanczos Interpolation genutzt.

Damit das Netz keine Daten auswendig lernen kann und dessen Performance Maße möglichst aussagekräftig sind, werden die Daten in Trainings-, Validierungs- und Testdaten unterteilt. Mit den Validierungsdaten wird das Netz während des Trainings getestet. Da man üblicherweise das Trainingsstop-Kriterium von den Validierungsergebnissen abhängig macht, benötigt man den dritten Testdatensatz, um das Netz mit anderen Netzen vergleichen zu können [35]. Trainiert man allerdings mehrere Netze und wählt auf den Testdaten aus, stellt dies wieder eine gewisse Abhängigkeit zu den Daten her. Daher sollte bei einem Machine Learning Wettbewerb ein vor der Teilnahme geheimer Datensatz zum Vergleich der Einreichungen benutzt werden.

Mit **Oversampling** werden Strukturen oder Klassen gezielt im Batch überrepräsentiert, was in RedLeaf durch das Stratified Patch Sampler Modul umgesetzt wird. Unterrepräsentierte Klassen benötigen länger zum trainieren, da deren Lerngradient auf nicht relevanter Information, also Rauschen, basiert. Kommen Klassen in manchen Batches garnicht vor, kann das Netz diese sogar verlernen, was es am Konvergieren hindert. Allerdings werden die Daten beim Oversampling redundant genutzt, wodurch auch ein zu kleiner Datensatz das Problem für eine nicht konvergierende Klasse sein kann.

Beim Normalisieren werden verschiedene Dimensionen auf ein gemeinsames Maß gebracht. Die verschiedenen Dimensionen sollten möglichst einheitliche Wertebereiche haben, soweit möglich auf gleichen Skalen sein und nicht zu große Werte an sich annehmen. Dabei sollten die Grenzen der benutzten Datentypen berücksichtigt werden. Zum Beispiel kann es nützlich sein, seine Daten auf einen Mittelwert μ von 0 und eine Standardabweichung σ von 1 zu bringen. Dabei nutzt man $y = \frac{x-\mu}{\sigma}$ für jede Feature Dimension einzeln [35]. Dieser Schritt wird in Variationen durch Normalisierungslayer auch im Netzwerk wiederholt (siehe Batch Norm in 3.1).

Die Bilder in einem Datensatz zeigen eine feste Perspektive. Allerdings gibt es viele Probleme in denen diese Eingaben leicht verändert sein könnten und trotzdem das selbe Ergebnis erwartet wird. Um solche Fälle

beim Training mit Einzufangen und damit das Netzwerk besser zu generalisieren, benutzt man Data Augmentation. Unter diesen Begriff fallen alle Veränderungen die man am Datensatz machen kann, ohne dadurch neue Label durch einen nicht automatisierten Prozess erzeugen zu müssen. Zum Beispiel beim Hinzufügen von Rauschen auf der Eingabe. Aber auch bei Rotation, Skalierung und Spiegelung muss man nur die selbe Operation auf dem Label durchführen. Allerdings können hierbei Füllwerte nötig sein.

3.3 Postprocessing

AUTOR*IN: NIKLAS AGETHEN

Im Bereich des Postprocessing haben wir den Ansatz der Erweiterung eines Segmentierers zu einer Kaskade mit einem Klassifikator betrachtet. Dabei arbeitet der Klassifikator auf den Ausgaben des Segmentierers und soll dessen Ergebnisse verbessern, wenn bspw. einzelne Klassen vom Segmentierer fehlerhaft segmentiert werden. Durch seine zusätzlichen Gewichte erweitert der Klassifikator die Komplexität des Gesamtmodells, kann sich speziell auf die fehleranfälligen Bereichen bzw. Klassen der Segmentiererausgabe konzentrieren und lernt diese im Idealfall zu korrigieren.

Der Ansatz, einen Segmentierer und einen Klassifikator in einer Kaskade zu verbinden, ist nicht neu. Dhungel et al. nutzen bspw. eine Kaskade aus mehreren CNNs und Random Forests zur Segmentierung von Brusttumoren [36]. Dabei werden zunächst mögliche Tumor-Kandidaten durch eine Form von Deep Neural Networks (DNNs) ermittelt, die Kandidaten dann von CNNs sowie Random Forests klassifiziert und abschließend mittels Connected-Components-Analyse verbunden. Dieses Vorgehen ermöglicht so das Erkennen und Ausschließen von falsch positiven Tumorkandidaten, u.a. durch die Verwendung von Random Forest Klassifikatoren.

Gegenüber DNNs bringen die in [36] verwendeten Random Forests den Vorteil mit sich, dass diese unabhängig von der Anzahl an Trainingsdaten weniger stark von Overfitting betroffen sind [37]. Da bei der Klassifikation jedes Bild des Trainingsdatensatzes als **ein** Trainingsdatum der jeweiligen Klasse zählt, fällt die notwendige Menge an Trainingsdaten an dieser Stelle schwerer ins Gewicht als bei einer Segmentierung, bei der jedes Pixel bzw. Voxel als Trainingsdatum angesehen werden kann. Dennoch erscheint auch die Verwendung eines DNN Klassifikators als eine mögliche Option für die Erweiterung eines Segmentierers zu einer Kaskade.

3.4 Netzarchitekturen

3.4.1 U-Net

AUTOR*IN: JANNES ADAM

In der medizinischen Bildverarbeitung ist eine Anforderung, dass neuronale Netze nicht nur klassifizieren, was auf einem Bild zu sehen ist, sondern auch wo es zu sehen ist. Außerdem stehen vergleichsweise wenig Trainingsdaten zur Verfügung, da für deren Annotation Fachpersonal benötigt wird.
Diese beiden Probleme versucht das **U-Net** zu lösen, welches zu den FCNs gehört und zur Segmentierung von Bildern gedacht ist. Die Idee dahinter ist, dass zusammengefasste Informationen auf niedriger Auflösung mit höheren Auflösungsstufen kombiniert werden, um eine hochauflösende Klassifikation zu erhalten. Dazu besteht das U-Net aus einem kontrahierenden Pfad (Downsampling), der jeweils mehrere Faltungen mit anschließender ReLU-Aktivierung, gefolgt von einem Pooling-Layer wiederholt. Auf jeder dieser Stufen wird die Anzahl der Feature-Kanäle verdoppelt. Dazu kommt ein expansiver Pfad (Upsampling), der genauso aufgebaut ist wie der kontrahierende Pfad, statt der Pooling-Layer jedoch Upsampling-Layer verwendet. Für das U-Net werden hier **Transposed-Convolutions** verwendet, die mittels einer Faltungsoperation die Auflösung verdoppeln.

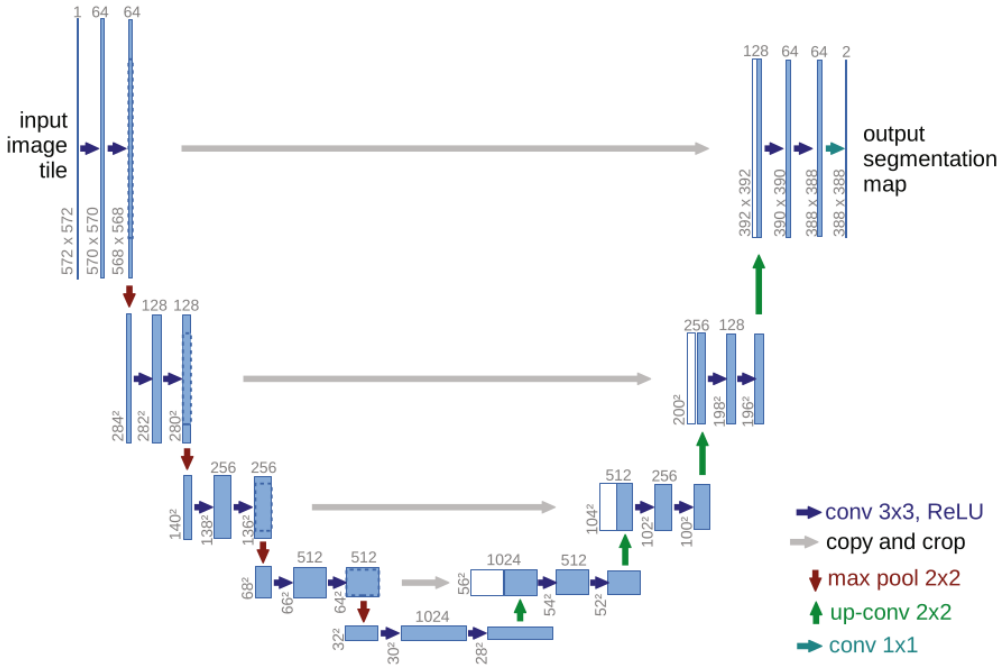


Abbildung 3.10: Visualisierung des U-Nets aus [38]

Zusätzlich existiert pro Stufe eine Querverbindung, die die Ausgabe der jeweils letzten Faltungsschicht einer Stufe als zusätzliche Kanäle mit den upgesampelten Informationen verbindet. Da das U-Net standardmäßig Unpadded-Convolutions verwendet, müssen die Daten auf der Querverbindung gecropped werden, um die gleiche Größe wie die Daten im Upsampling-Path zu erreichen. Zum Schluss wird durch eine $1*1$-Faltung die Anzahl der benötigten Klassen erreicht. Durch die durch den Aufbau entstehende U-Form, erhielt das Netz den Namen U-Net.
Wichtig ist, dass im Upsampling-Path genügend Kanäle zur Verfügung stehen, um die Kontextinformationen zu behalten. Da im Original keine Padded-Convolutions verwendet werden, werden die Randbereiche des Bildes nicht segmentiert. Um dieses Problem zu beheben, werden die Eingabebilder durch Spiegelung der Randbereiche gepaddet. Um ein Überlaufen des Hauptspeichers zu verhindern werden große Bilder nach der **Overlap-Tile-Strategy** aufgeteilt. Dazu werden aus dem Bild Kacheln ausgeschnitten, die sich mit dem für das Padding nötigen Bereich überlappen, sodass das gesamte Bild segmentiert werden kann.
Durch die Unpadded-Convolutions muss bei der Wahl der Größe der Bilder und deren Padding darauf geachtet werden, dass die Kombination für

Level	Input Size	Minimum Output Size	Padding	Valid Tile Size Offset
2	18x18	2x2	8x8	2x2
3	44x44	4x4	20x20	4x4
4	92x92	4x4	44x44	8x8
5	188x188	4x4	92x92	16x16

Tabelle 3.2: Valide Eingabegrößen für das U-Net

die gewählte Architektur valide ist. In Tabelle 3.2 sind die Kombinationen für die Standardarchitekturen dargestellt.
Bei der Auswahl der Anzahl der Level spielt das rezeptive Feld eine große Rolle, da dieses angibt, welchen Bereich des Bildes das U-Net pro zu klassifizierendem Voxel sieht. Tabelle 3.3 gibt eine Überblick darüber, wie groß das rezeptive Feld in Abhängigkeit von Leveln und Filtergröße ist [38].

Filter Size	Levels			
	2	3	4	5
2	10	22	46	94
3	18	44	92	188
4	26	62	138	282

Tabelle 3.3: Rezeptives Feld des U-Nets

3.4.2 AU-Net

AUTOR*IN: MARKUS RINK

Das Attention U-Net ist die Erweiterung der U-Net Architektur um Attention Gates [39, 40]. Außerdem können 2D und 3D Filter pro Level eingestellt werden, wodurch anisotrope rezeptive Felder ermöglicht werden.

Der Anstieg des Volumens vergrößert sich mit jeder zusätzlichen Dimension, was auch als „Fluch der Dimensionalität“ bezeichnet wird [41]. Möchte man Strukturen mit einer gewissen Länge mit dem rezeptiven Feld abdecken, so müsste ein 3D Netz exponentiell größer sein, als ein 2D Netz. Der Mensch bewegt sich in drei Raumdimensionen, weswegen

es sinnvoll sein kann, auch 3D Netzwerke für Probleme mit diesem Kontext zu benutzen. Es könnte sein, dass 2D Netzwerke den 3D Kontext nicht ideal annähern können. Anisotrope Netze verringern die Voxelanzahl in gewählten Dimensionen, wodurch der Volumenanstieg gegenüber einer 2D Eingabe frei gewählt werden kann.

Attention Gates spielen mit dem Namen auf die Attention Mechanismen von Transformern an, welche in der Sprachverarbeitung eingesetzt werden [42]. Die Rechenleistung dieses Mechanismus steigt exponentiell mit der Länge der betrachteten Sequenz an Wörtern, in der Bildverarbeitung entsprechend Sequenzen von Pixeln. Die Größe von Bildern macht dieses Verfahren im Allgemeinen zu teuer für die Bildverarbeitung. AGs sollen dabei eine leichtgewichtige Variation für CNNs darstellen.

Attention Gates erzeugen eine Gewichtung für die Skip Connection. Diese Gewichtung soll wichtige Bereiche für die Segmentierung hervorheben, weshalb man sie Salienz Karten nennt. Salienz ist ein zu hervorhebendes Merkmal. Das Netz soll in diesen Schichten lernen, bestimmte Regionen zu priorisieren und so ein besseres Ergebnis erzielen. Die Salienz Karten sind im Idealfall die Segmentierungen, welche das Netz ausgeben soll. Daher kann man diese Layer direkt mit dem Target Label vergleichen, wobei das Layer dem Loss hinzugefügt wird. Das Berechnen von Losses auf Layern im Netz nennt sich Deep Supervision und ist in einer Variation in RedLeaf verfügbar. Die Implementierung von Deep Supervision im AU-Net erzeugt eine Verbindung vom Gating Signal zum Output, anstatt vom α Layer zum Output. Ozan Oktay et al. schlagen vor, ein Attention Gate für jedes Label zu benutzen [40].

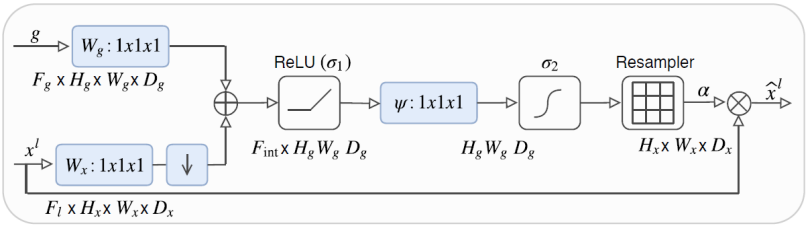


Abbildung 3.11: Attention Gate

Attention Gated Networks: Learning to Leverage Salient Regions in Medical Images [40]

Das Attention Gate (vgl. Abbildung 3.11) bekommt als Eingabe die Skip Connection des U-Nets als x^l und die Ausgabe des letzten Layers des unteren Levels, welches hier Gating Signal g genannt wird. Diese werden jeweils mit den Gewichten W_x und W_g transformiert. Da die Skip Connection ein größerer Tensor ist, wird dieser erst auf die Größe des Gating Signals runter skaliert. Danach werden diese Tensoren addiert und mit ReLU abgebildet. ψ transformiert nun die Kombination aus Gating Signal und Skip Connection und ist das dritte Layer, welches im Attention Gate trainiert werden muss. Nach einer zweiten nicht-Linearität, in diesem Fall Sigmoid, muss wieder auf die Auflösung der Skip Connection skaliert werden, wonach man die Salienz Karte α bekommt.

3.4.3 U-ResNet

AUTOR*IN: NIKLAS AGETHEN

Die U-ResNet Architektur erweitert das in Kapitel 3.4.1 beschriebene U-Net um Residualblöcke, die das Trainieren und Optimieren von neuronalen Netzen mit vielen Schichten verbessern [43]. Die Residualblöcke wurden erstmalig in der ResNet Architektur von He et al. [43] beschrieben. Die Autor*innen zeigen darin, dass das Hinzufügen von weiteren Convolutionschichten zu einem DNN nicht zwangsläufig zu einem kleineren Loss bzw. einer höheren Accuracy führt. Viel mehr wird dort das Phänomen beschrieben, dass tiefere neuronale Netze schlechter lernen als ihre kleineren Pendants. Dabei unterscheiden sich die Netze nur in der Anzahl an Schichten, alle anderen Parameter sind identisch. Da das Ergebnis eines kleineren Netzes eine Teilmenge des Ergebnisses größerer Netze darstellt (das größere Netz bildet in den zusätzlichen Schichten lediglich die Identitätsfunktion ab und reicht so das Ergebnis der früheren Schichten durch), begründen die Autor*innen das schlechtere Abschneiden größerer neuronaler Netze durch das schwierigere Optimieren. Daher soll dem ResNet das Lernen der Identitätsfunktion erleichtert werden, indem die Autor*innen Shortcut-Verbindungen um die Convolutionschichten einführen (siehe Abbildung 3.12).

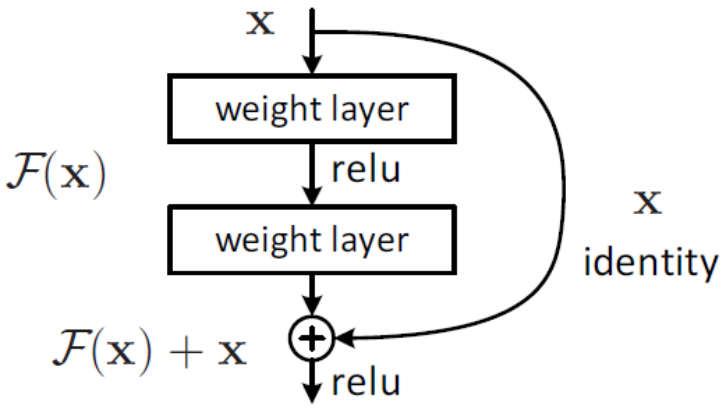


Abbildung 3.12: Residualblock bestehend aus Residuum und Shortcut-Verbindung [43]

Durch diese Shortcut-Verbindungen werden die Eingabedaten einer Convolutionschicht zu den Ausgabedaten addiert und stellen somit die Identitätsfunktion dar. Durch diese Konstruktion soll verhindert werden, dass bereits Gelerntes wieder „vergessen“ bzw. „verlernt“ wird. Wenn die Identitätsfunktion, d.h. die Shortcut-Verbindung, bereits die optimale Lösung abbildet, muss im Extremfall das neuronale Netz nun lernen die Gewichte des Residuums, d.h. die vom Shortcut umgangene(n) Schicht(en), auf Null zu setzen. Das von He et al. beschriebene ResNet stellt eine Erweiterung des VGG Netzes von Simonyan und Zisserman [44] dar, indem im Wesentlichen nur die Convolutionschichten durch Residualblöcke ersetzt werden. Mit diesem neuronalen Netz belegten die Autor*innen den ersten Platz bei der ILSVRC 2015 Klassifikations-Challenge sowie der COCO 2015 Detektions-Challenge.

Das U-ResNet orientiert sich an der Architektur des U-Nets, verwendet allerdings die erwähnten Residualblöcke anstelle von Convolutionschichten. Da das U-ResNet eine zusätzliche Convolutionschicht zu Beginn des Netzes einführt, erweitert sich das rezeptive Feld der Architektur gegenüber des U-Nets (siehe Tabelle 3.4). Die Tabelle vergleicht die Größe des rezeptiven Felds unterschiedlicher Levelmengen und Kernelgrößen mit der festen Anzahl von zwei Convolutionschichten pro Level.

Durch die Verbindung von U-Net und ResNet werden die für die medizinische Domäne konzipierten Ansätze des U-Nets (siehe 3.4.1) mit dem

filter size	levels			
	2	3	4	5
2	11	23	47	95
3	20	46	94	190
4	29	65	141	285

Tabelle 3.4: Rezeptives Feld des U-ResNets

für tiefere Netze gedachten Konzepts des ResNets gepaart und dies verspricht vor allem verbessertes Training von tiefen U-Nets. Dieser Ansatz wurde bereits in anderen Werken angewendet, bspw. beschreiben Zioulis et al. [45] in ihrem Paper eine U-ResNet Architektur zur Generierung von Depth Masks zu gegebenen Eingabebildern.

3.4.4 DeepMedic

AUTOR*IN: MARCEL PLUTAT

Deep Medic ist ein 3D CNN, welches von Kamnitsas et al. im Rahmen der BRATS Challenge entworfen wurde [46]. Die Aufgabe der BRATS Challenge ist es Hirntumore sowie Läsionen und andere Merkmale in einem MRT des Gehirns zu erkennen.
Die Architektur von Deep Medic besteht aus zwei Pfaden mit unterschiedlichen Auflösungen. Der „niedrig“ aufgelöste Pfad ist ein grober Kontextbereich der downgesampled wird. Der „normal“ aufgelöste Pfad ist aus der Mitte dieses Kontextes ausgeschnitten. Der Low Resolution Pfad wird abschließend hochgesampled, um mit der normalen Auflösung konkateniert zu werden. Deep Medic besteht aus elf Schichten mit jeweils einem $3 \times 3 \times 3$ Kernel in den Convolutionalschichten. Die Architektur besteht aus acht Convolutionalschichten gefolgt von zwei Fully Connected Schichten, die durch $1 \times 1 \times 1$ Convolutions umgesetzt sind, und einer abschließenden Klassifikationsschicht. Die kleinen Kernel wurden verwendet, da die Autor*innen zeigen konnten, dass mehrere kleinere Convolutions äquivalent zu einer großen Convolution sind, dabei aber wesentlich weniger Gewichte pro Layer benötigen [46].

Die Erweiterung von Deep Medic um Residual Connections stammt ebenfalls von Kamnitsas et al. [47]. Die Residual Verbindungen für Deep Medic umfassen jeweils zwei Convolutionalschichten. Eine weitere Änderung ist, dass die Reihenfolge der Operationen pro Block auf Batch Normalization, Nicht-Linearität und abschließend die Convolution geändert wurde. Die Autor*innen haben gezeigt, dass diese Reihenfolge die besten Ergebnisse erzielt [47].

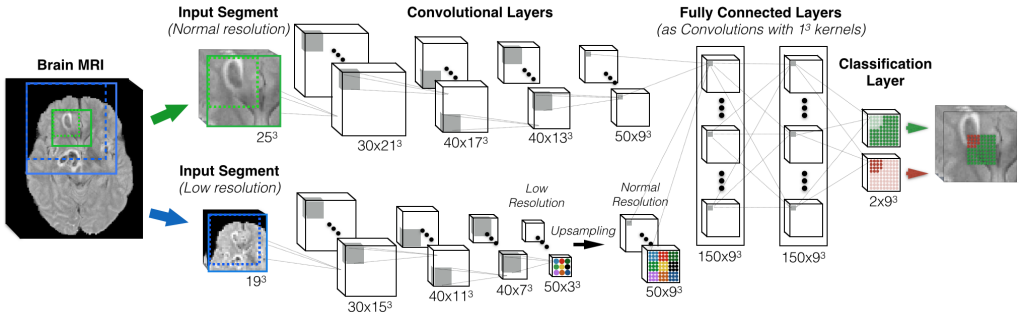


Abbildung 3.13: Deep Medic Architektur aus [48]

Abbildung 3.13 zeigt die Netzarchitektur von Deep Medic, wobei zur Veranschaulichung in dieser 5^3 Kernel in den Convolutionalschichten verwendet wurden. Die „echte“ Deep Medic Architektur verwendet 3^3 Kernel in den Convolutions und hat stattdessen doppelt so viele Convolutionalschichten.

3.4.5 nnU-Net

AUTOR*IN: LENA PHILIPP

Das nnU-Net („no new-Net“) Framework wurde 2018 von Isensee et al. vorgestellt [49]. Die Grundidee basiert auf dem Erfolg des U-Nets und „der Systematisierung, des komplexen Prozesses der manuellen Methodenkonfiguration“. Dabei wird keine neue Architektur oder grundlegende Architekturanpassung vorgeschlagen, sondern das U-Net verwendet. Bei der KiTS19 Challenge zeigte sich, dass unter den 15 besten Einreichungen alle auf dem U-Net basierten. In der Auswertung der Netze zeigt sich, dass die Methodenkonfiguration ausschlaggebend für den Erfolg ist und

nicht Variationen bei der Architektur. Die automatische Konfiguration des

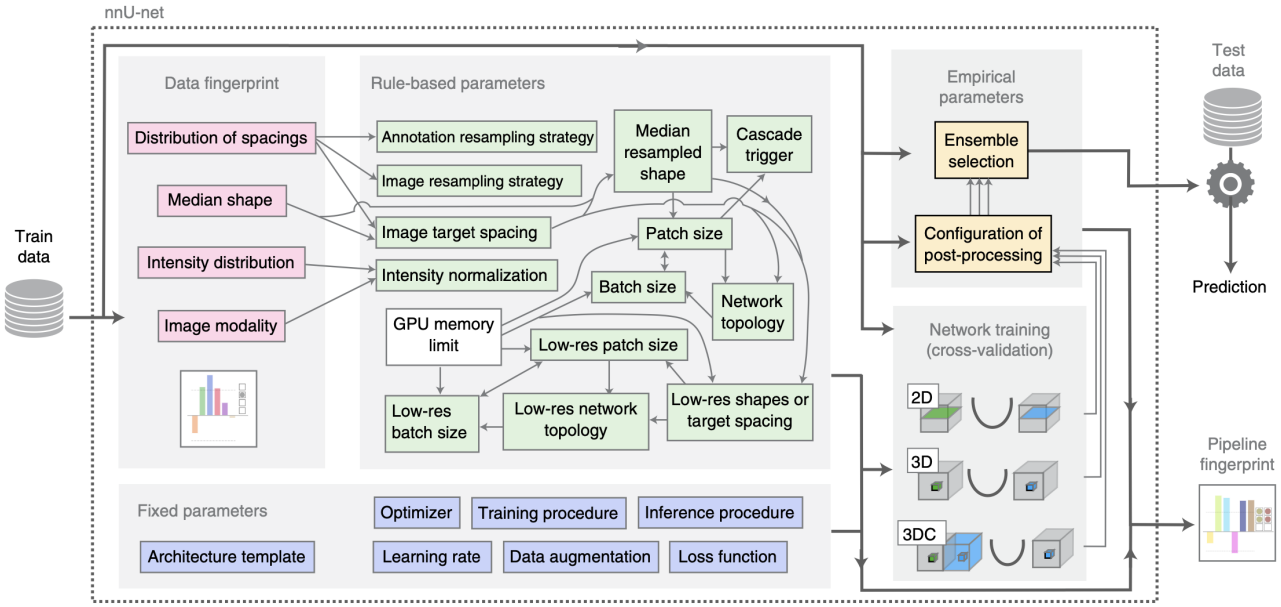


Abbildung 3.14: nnU-Net. Die auf den Daten basierenden Einstellungen werden pink dargestellt, die regelbasierten grün, die festen blau und die empirischen gelb [49].

nnU-Nets bezieht sich auf den gesamten Trainingsprozess (Vorverarbeitung, Architektur, Training, Nachverarbeitung). Als Basis wird der Datensatz sowie feste und regelbasierte Parameter genutzt.
In der Analyse verschiedener Datensätze zeigte sich, dass jeder Datensatz einen eigenen Fingerprint trägt, der beachtet werden muss um die optimalen Trainingseinstellungen finden zu können, wie die Voxelgröße und das Verhältnis der Klassen. Basierend auf diesem Fingerprint werden die Entscheidungen getroffen, die in Abbildung 3.14 in Grün aufgelistet werden. Die darin gezeigten Pfeile stellen Abhängigkeiten zwischen den Parametern dar.
Ein Beispiel für eine Regel, die als Grundlage für Entscheidungen dienen, ist, dass jede Batchgröße größer als eins zu einem robusten Training führt. Ein anderes Prinzip, nach dem Parameter festgelegt werden, ist, dass die Tiefe so gewählt wird, dass das rezeptive Feld mindestens so groß wie die Patch Größe ist um Verlust an kontextuellen Informationen zu vermeiden. Diese Regeln lassen sich einfach erweitern. Diese Regeln legen also die Konfiguration des Netzes und des Trainingsprozesses fest. Schließlich werden bis zu drei Konfigurationen (2D, 3D und 3D Kaskade)

trainiert, über eine fünffache Kreuzvalidierung ausgewertet und ein Ensemble dieser Modelle ausgewählt. Die empirischen Regeln entscheiden über den Einsatz von Nachverarbeitung [49].
Bei einer Vielzahl von internationalen Challenges (darunter Multimodal Brain Tumor Segmentation Challenge 2018 - 2. Platz, Automated Cardiac Diagnosis Challenge - 1.Platz, KiTS - 1.Platz) wurde mit diesem Ansatz erfolgreich teilgenommen [50].

3.5 Evaluation

AUTOR*IN: TIMO GÜNNEMANN

Nachdem ein Modell trainiert wurde, ist es interessant zu sehen, wie gut es gegenüber anderen Modellen funktioniert. Dafür führt man im ersten Schritt eine Inferenz auf dem Testdatensatz durch. Eine Inferenz nutzt die Ausgabe des trainierten Modells und wendet sie auf dem Testdatensatz an. Nachdem die Inferenz ausgeführt wurde, folgt die Evaluation anhand der Evaluationsmaße. Dazu vergleicht man das Ergebnis von der Inferenz mit den Referenzdaten. Die Evaluationsmaße geben jeweils an, wie gut die Strukturen in dem Bild segmentiert wurden. Oft handelt es sich dabei um einen Wert zwischen 0 und 1. Ein Beispiel für ein Evaluationsmaß ist der Sørensen-Dice-Koeffizient. In dem Fall gilt, umso höher der Wert liegt, desto besser wurde die Struktur segmentiert und umso niedriger der Wert, desto schlechter wurde die Struktur segmentiert. Nach der Evaluation ist es möglich verschiedene Modelle anhand dieser Evaluationsmaße zu vergleichen. Voraussetzung ist dafür, dass alle Modelle anhand der gleichen Maße evaluiert wurden.

Es gibt verschiedene Evaluationsmaße, die alle zwar unterschiedlich sind, aber dem gleichen Zweck dienen. Es wird hier auf ein paar grundlegende Maße eingegangen, die für das Verständnis der nächsten Kapitel erforderlich sind.

3.5.1 Konfusionsmatrix

Die Konfusionsmatrix beschreibt die Performanz des Modells bei einem Klassifikationsproblem, indem man die vorhergesagten Werte des Modells bzw. die Ausgabe und die Referenzdaten gegenüber stellt. Der grundlegende Aufbau ist in der Abbildung 3.15 zu sehen. Anstatt von TP, TN usw. stehen in den Feldern die entsprechenden Zahlenwerte. In diesem Fall sind es nur zwei Klassen, dadurch ergeben sich TP, TN, FP und FN. Es gibt aber auch Konfusionsmatrizen für mehr als zwei Klassen. Wenn das der Fall ist, existieren die TP, TN, FP und FN Werte nur klassenweise. Konfusionsmatrizen mit mehr als zwei Klassen sind oft farbkodiert, für eine bessere Übersichtlichkeit. Die Vorhersage ist dann optimal, wenn für alle Klassen auf der Hauptdiagonalen Werte > 0 stehen und auf allen anderen Feldern Werte = 0 [51].

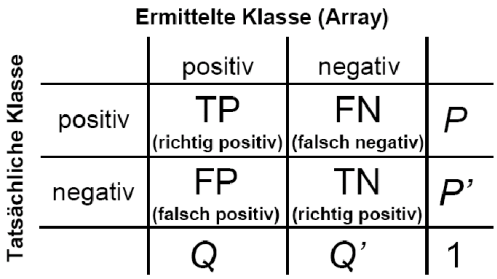


Abbildung 3.15: Aufbau Konfusionsmatrix [52]

- **True Positive** (TP): Der vorhergesagte Wert und der Referenzwert sind beide positiv.
- **True Negative** (TN): Der vorhergesagte Wert und der Referenzwert sind beide negativ.
- **False Positive** (FP) oder Fehler 1. Art: Der vorhergesagte Wert ist positiv und der Referenzwert ist negativ.
- **False Negative** (FN) oder Fehler 2. Art: Der vorhergesagte Wert ist negativ und der Referenzwert ist positiv.

Mit der Konfusionsmatrix lassen sich die folgenden Maße berechnen.

- **Accuracy** gibt an, wie viele von allen vorhergesagten Werten

korrekt vorhergesagt wurden.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

- **Precision** gibt an, wie viele von allen positiv vorhergesagten Werten korrekt als positiv vorhergesagt wurden.

$$Precision = \frac{TP}{TP+FP}$$

- **Recall/Sensitivity** gibt an, wie viele von allen tatsächlich positiven Werten korrekt als positiv vorhergesagt wurden.

$$Recall/Sensitivity = \frac{TP}{TP+FN}$$

- **True Negative Rate/Specificity** gibt an, wie viele von allen tatsächlich negativen Werten korrekt als negativ vorhergesagt wurden.

$$Specificity = \frac{TN}{TN+FP}$$

- **False Positive Rate** gibt an, wie viele von allen tatsächlich negativen Werten fälschlicherweise als positiv vorhergesagt wurden.

$$FalsePositiveRate = \frac{FP}{TN+FP}$$

Bei den folgenden Maßen ist X die vorhergesagte Menge oder die Ausgabe des Modells und Y die Referenzmenge. Der Dice-Koeffizient und Jaccard-Index geben jeweils an, wie ähnlich zwei Mengen zueinander sind. Das bedeutet, die Ausgabe des Modells wird mit der Referenzmenge verglichen. Am Ende kommt ein Wert zwischen 0 und 1 raus, wobei 0 keine Überschneidung und 1 komplette Überschneidung bedeutet.

3.5.2 Sørensen-Dice Koeffizient

Der **Dice Koeffizient** ist wie folgt definiert:

$$DSC = \frac{2 * |X \cap Y|}{|X| + |Y|}$$

Das bedeutet für zwei Bilder oder Volumen, die verglichen werden sollen, dass die Anzahl der Pixel/Voxel, die in beiden Bildern zu derselben Klasse gehören, gezählt werden und durch die Anzahl der Pixel/Voxel dieser Klasse in beiden Bildern geteilt werden. Dieser Wert wird mit zwei multipliziert. Daraus ergibt sich für das Binärproblem, die folgende Formel [53].

$$DSC = \frac{2TP}{2TP + FP + FN}$$

3.5.3 Jaccard Index

Der **Jaccard Index** lässt sich folgendermaßen berechnen:

$$J = \frac{|X \cap Y|}{|X \cup Y|}$$

Auf Bilder übertragen heißt das, dass die Anzahl der Pixel/Voxel, die in beiden Bildern/Volumen zu derselben Klasse gehören, durch die Anzahl der Pixel/Voxel dieser Klasse in beiden Bildern geteilt werden, wobei im Divisor Pixel/Voxel, die in beiden Bildern dieser Klasse zugeordnet sind, nur einfach gezählt werden [54].

3.5.4 Surface Dice

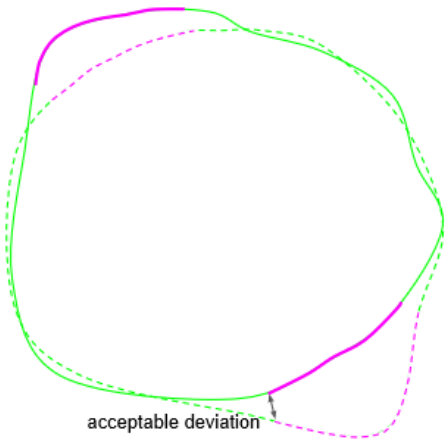


Abbildung 3.16: Überlappung der Oberflächen

Der **Surface Dice** unterscheidet sich in sofern vom Dice, dass er nicht die volumetrische Überlappung betrachtet, sondern die Überlappung der Oberflächen von zwei Strukturen innerhalb eines Toleranzbereiches. Es wird somit der Abstand zwischen der Oberfläche der vorhergesagten Struktur - in Abbildung 3.16 als grüne durchgezogene Linie dargestellt - mit der Oberfläche der Referenz - der grünen gestrichelten Linie - verglichen. Wenn der Abstand innerhalb eines bestimmten Toleranzbereiches liegt, zählt das zum guten Oberflächenteil. Dieser wird dann mit der gesamten Oberfläche verglichen, also die Summe der vorhergesagten Oberfläche und der Oberfläche der Referenz [55].

4 Methoden

4.1 MeVisLab

4.1.1 Verwendete Module und typische Einstellungen

AUTOR*IN: TINGTING XUE

Abbildung 4.1 demonstriert ein schematisches Diagramm der Hauptkomponenten des MevisLab-Netzwerks, einschließlich jedes Moduls und der Beziehung zwischen ihnen, worin die Daten sich von unten nach oben durch die Module bewegen, während Abbildung 4.2 die Parametereinstellungsdialogfelder jedes Moduls zeigt.

Das Modul *LoadDataIntoStreams* (siehe Abbildung 4.2(A)) wird verwendet, um entsprechende Bilder zu laden und dem Original-Stream sowie den Labels-Stream für die Remote-Datenbereitstellung bereitzustellen. Diese Stream-Objekte stellen die gesamten Daten bereit, während die Module selbst Ausschnitte daraus laden können.

Danach wird das Modul *ResampleImageStreams* (siehe Abbildung 4.2(D)) für die Vorverarbeitung verbunden, um Bildströme in ein gemeinsames, normalisiertes Koordinatensystem zu bringen. Hier wird die Voxelsize von den x-, y- und z-Achse anfänglich jeweils auf $2,5mm$ eingestellt und kann nachträglich angepasst werden. Die Ansichtsrichtung soll für das Ausgabenziel auf Transversal gesetzt werden.

Dann folgt das Modul *CacheStreamImages* (siehe Abbildung 4.2(C)), um alle Bilder im .mlimage-Dateiformat am angegebenen Cache-Pfad unter Gaia (siehe Abschnitt 2.2.6) zu speichern.

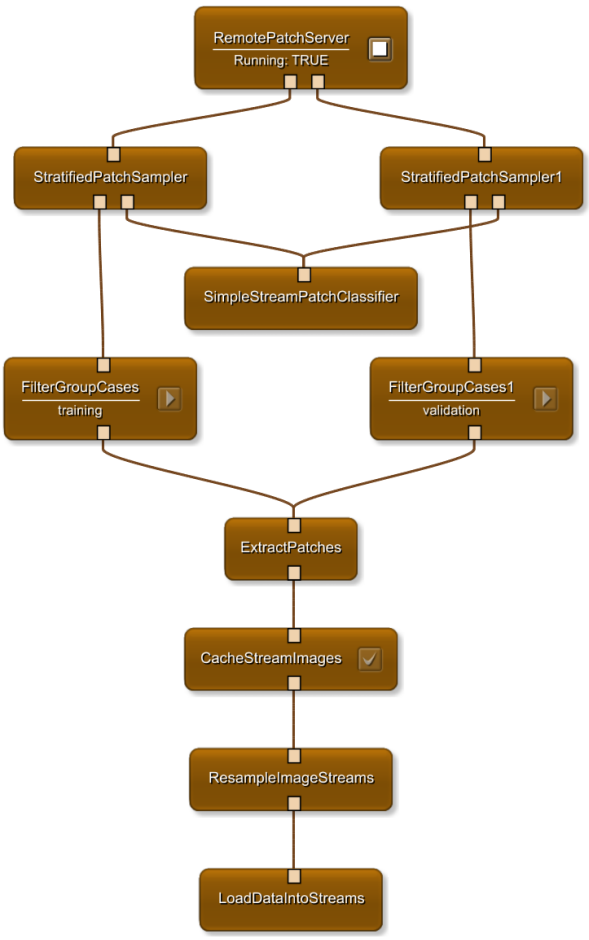


Abbildung 4.1: MeVisLab Netzwerke. Die Daten bewegen sich von unten nach oben durch die Module.

Als Nächstes wird das Modul *ExtractPatches* (siehe Abbildung 4.2(B)) verbunden, welches einen Stream von seiten-basierten Bildern nimmt, daraus Patches extrahiert und ein Stream des Patches erzeugt. Hier werden kleinere Label-Patches als Eingabe-Patches wegen des Valid-Paddings benötigt.

Am Ausgang des Moduls *ExtractPatches* werden zwei *FilterGroupCases* Module (siehe Abbildung 4.2(G)) gleichzeitig angeschlossen, um die gestreamten Fälle jeweils auf die in der Trainings- und Validierungsgruppe definierten Fallindizes einzuschränken.

Wenn nötig, werden die beiden *FilterGroupCases* Module jeweils mit einem *StratifiedPatchSampler* Modul (siehe Abbildung 4.2(F)) verbunden, welche Oversampling durch einen regulären Ausdruck ermöglicht.

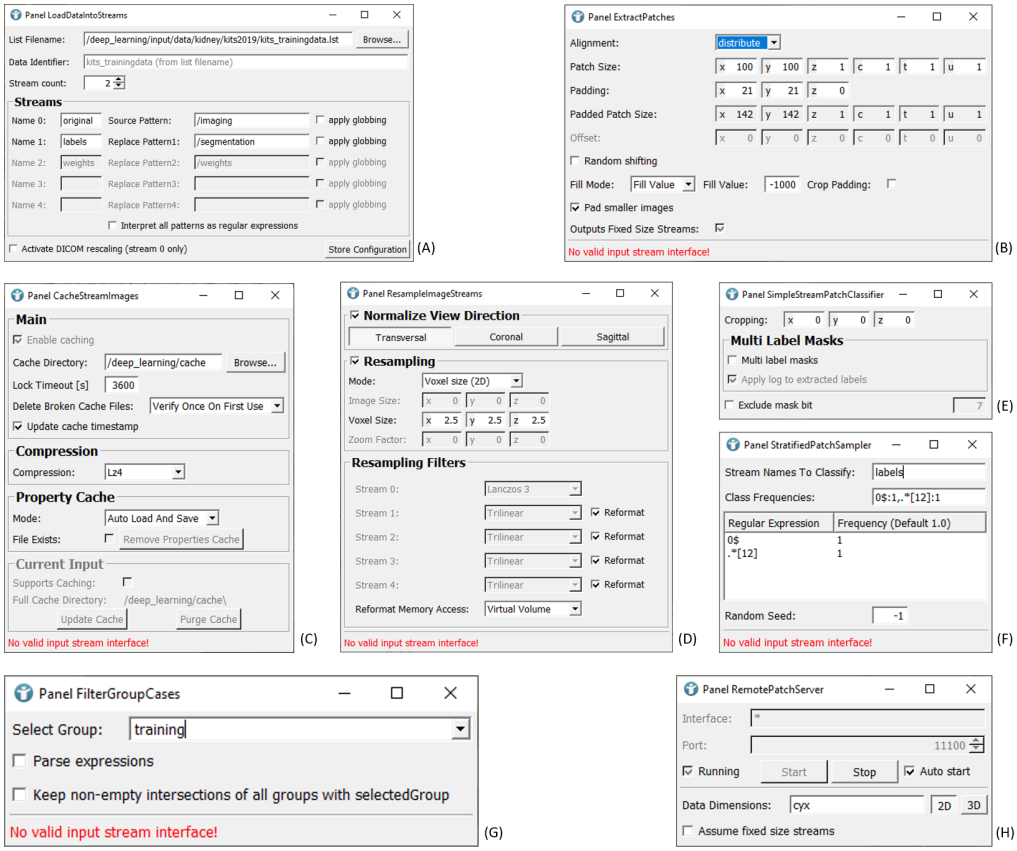


Abbildung 4.2: Dialogfelder für jedes Modul in den MeVisLab Netzwerke. (A) Modul *LoadDataIntoStreams*; (B) Modul *ExtractPatches*; (C) Modul *CacheStreamImages*; (D) Modul *ResampleImageStreams*; (E) Modul *SimpleStreamPatchClassifier*; (F) Modul *StratifiedPatchSampler*; (G) Modul *FilterGroupCases*; (H) Modul *RemotePatchServer*.

In diesem Fall wird ein *SimpleStreamPatchClassifier* Modul (siehe Abbildung 4.2(E)) mit beiden *StratifiedPatchSampler* Modulen gleichzeitig verbunden, was ermöglicht, beim Training eines neuronalen Netzes die Zusammensetzung von Training-Mini-Batches und Validierungsdaten auf der Client-Seite zu konfigurieren.

Abschließend werden der Trainings- und Validierungsstream jeweils mit den entsprechenden Schnittstellen des Moduls *RemotePatchServer* (siehe Abbildung 4.2(H)) verbunden.

4.1.2 PreprocessingClone

AUTOR*IN: JANNES ADAM

Damit die Ergebnisse bei der Inferenz nicht unnötig schlechter werden, müssen die Daten genauso vorbereitet werden, wie sie auch für das Training vorbereitet wurden. Um dieses für anwendende Personen zu vereinfachen, war die Idee für den PreprocessingClone ein Modul für das Inferenz-MeVisLab-Netzwerk zu entwickeln, dass das Preprocessing aus dem zugehörigen Trainingsnetzwerk kopiert.
Um den PreprocessingClone zu verwenden, muss im Trainings-Netzwerk eine Gruppe von Modulen markiert werden, die für das Preprocessing zuständig ist und die kopiert werden soll. In der GUI muss nur der Pfad zur Trainingsdatei und der Name der Gruppe angegeben werden. Das Modul kopiert das markierte Netzwerk und bildet es intern nach. Dazu werden alle Module wie im Vorbild benannt, um die Verbindungen und Parameter korrekt übernehmen zu können. Da das Trainingsnetzwerk jedoch mit Streams arbeitet und das Inferenz-Netzwerk mit Einzelbildern, wird das Bild in einen Stream konvertiert und dieser Stream an das geklonte Netzwerk angeschlossen. Umgekehrt wird der Ausgabestream in ein Bild konvertiert, um es an das Inferenz-Netzwerk weitergeben zu können. Damit dieses Vorgehen funktioniert, muss die definierte Gruppe genau einen Eingabe- und einen Ausgabestream besitzen.
Zusätzlich werden Server Fields unterstützt, die das Ändern von Parametern in dem Trainingsdashboard erlauben. Wenn ein Parameter über das Cluster einstellbar ist, wird dieser in der log.pkl-Datei angeben. Daher kann der*die Anwender*in optional einen Pfad zu dieser Datei angeben. In diesem Fall werden alle Felder, zu denen ein Server Field angegeben wurde, mit den verwendeten Werten überschrieben.

4.1.3 Case Visualization

AUTOR*IN: JANNES ADAM

Die Visualisierungstools von MeVisLab werden dazu genutzt, die Ergebnisse der neuronalen Netze zu visualisieren, um eventuelle Schwachstellen entdecken zu können. Um dieses einfach zu ermöglichen, besitzt Challengr eine Schnittstelle zu MeVisLab, mit der Ergebnisse von zwei Sessions verglichen werden können. Dazu werden in Challengr zwei Sessions für den Vergleich ausgewählt und ein Case angegeben, der visualisiert werden soll. Diese Informationen werden an MeVisLab geschickt, wo das zugehörige Bild und die Segmentierungsergebnisse geladen werden.

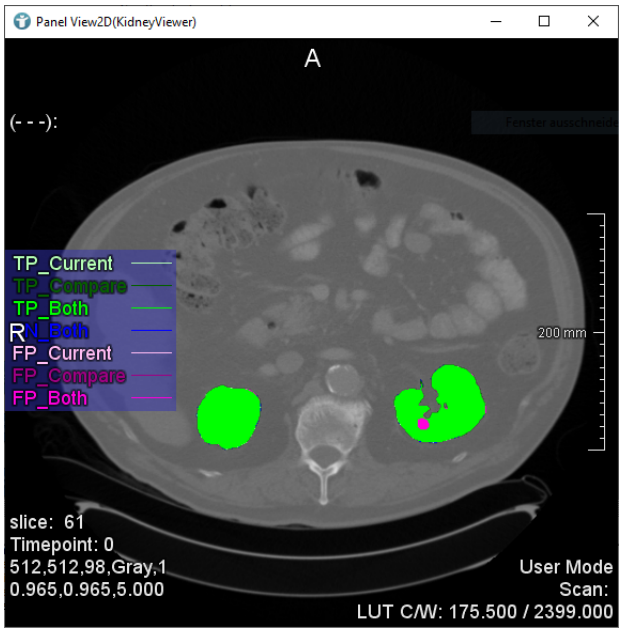


Abbildung 4.3: Beispielhafte Visualisierung der Segmentierung einer Niere durch zwei neuronale Netze. In diesem Fall haben beide Netze den Großteil der Niere richtig segmentiert. In der rechten Niere jedoch, haben beide Netze einen Bereich als Niere markiert, der keine Niere sein sollte.

Für die Challenges wurde eine Visualisierung gewählt, die vergleichend anzeigt, wie die Bilder von den Netzen segmentiert wurden. Dazu existiert ein Viewer pro zu segmentierender Klasse. Dieser zeigt von beiden Sessions an, welche Bereiche korrekt segmentiert, falsch positiv oder

falsch negativ bewertet wurden. Ein Beispiel ist in Abbildung 4.3 zu sehen.
Dazu wurde ein Macromodul erstellt, welches diese Informationen aus den Bewertungen und der Annotation zusammenfasst. Dieses geschieht mit Arithmetic-Modulen, die für die beiden Sessions, die TP-, FP- und FN-Bereiche berechnet (siehe Abb. 4.4). Diese Ergebnisse werden so kombiniert, dass jede mögliche Kombination der Klassifizierung beider Sessions in eine diskrete Zahl codiert wird. Die Zahlen werden in Farben abgebildet und daraus das Overlay für das Eingabebild erstellt.

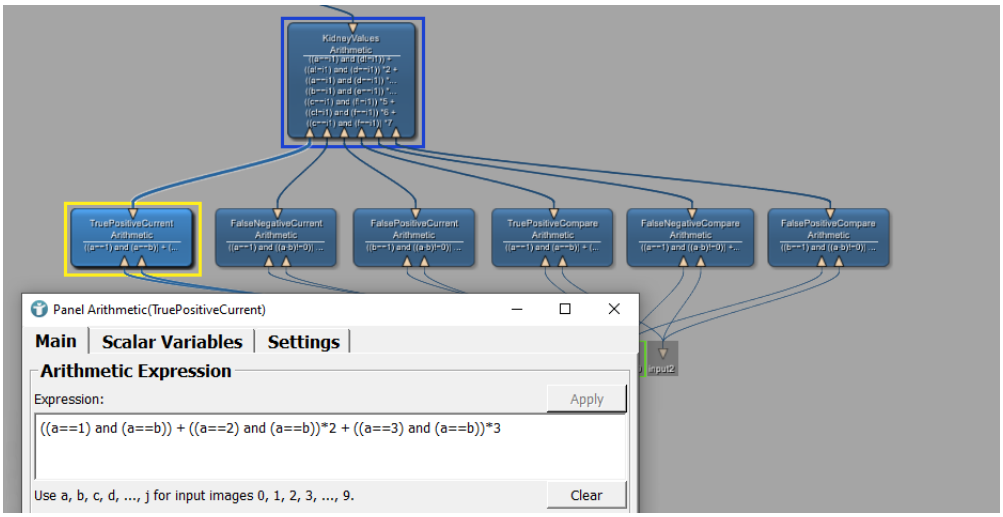


Abbildung 4.4: Ausschnitt des Macromoduls zur Klassifizierung der Bereiche in TP , FN und FP: Es gibt je ein Arithmetic-Modul pro Klasse und pro Session. Sie erhalten jeweils die Segmentierung und Annotation und vergleichen auf (Un-)Gleichheit und codieren die Ergebnisse. Diese werden vom oberen Arithmetic-Modul weiterverarbeitet.

4.2 RedLeaf

4.2.1 U-Net

AUTOR*IN: TINGTING XUE

In RedLeaf war bereits die U-Net-Modellierung mit TensorFlow [56] und Keras [57] implementiert. Es besteht jeoch ein Konflikt zwischen der *Tens-*

orFlow-Implementierung und der Challengr-Inferenz, was zu einer sehr geringen Effizienz führt. Deswegen haben wir die U-Net-Struktur mit *Keras* in der weiteren Aufgaben verwendet.

Im Folgenden eine Liste der von uns betrachteten Parameter bei der Verwendung vom U-Net:

- *levels*: die Anzahl der Ebene in der U-Net-Architektur (Standardwert: **5**)
- *filter_size*: die Größe des Convolution-Kernels (Standardwert: **(3, 3)**)
- *normalization*: **'none'**, **'batch_normalization'**, **'group_ normalization'** oder **'self_normalizing'** (Standardwert: **'none'**)
- *dropout*: ob Dropout verwendet wird (Standardwert: **False**)
- *input_channels* (Standardwert: **1**)
- *output_channels* (Standardwert: **2**)
- *base_filters*: die Anzahl der ursprünglich generierten Feature Maps (Standardwert: **64**)
- *conv_mode*: Convolution-Typ. **'valid'** bedeutet, dass zwischen Convolutional Layers kein Padding verwendet wird; **'same'** bedeutet, dass zwischen Convolutional Layers Padding verwendet wird, sodass Ausgabe und Eingabe die gleiche Größe haben (Standardwert: **'valid'**)
- *nonlinearity*: in der Layer verwendete Aktivierungsfunktion; bei *normalization* = **'self_normalizing'** hat dieser Parameter keine Auswirkung, da SeLu-Aktivierungsfunktion verwendet wird (Standardwert: **'relu'**)
- *final_layer_nonlinearity*: finale Aktivierungsfunktion (Standardwert: **'relu'**)
- *softmax*: ob die Aktivierungsfunktion Softmax verwendet wird (Standardwert: **True**)
- *fixed_input_shape*: Erstellen einer Formergänzung für feste Eingabesform in der Input Layer (Standardwert: **None**)

- *activation_after_batchnorm*: ob die Aktivierungsfunktion nach der Batch-Normalisierung ausgeführt wird; bei *normalization* = **'self_ normalizing'** hat dieser Parameter keine Auswirkung, da SeLu- Aktivierungsfunktion verwendet wird (Standardwert: **True**)
- *normalization_options* (Standardwert: **None**):
  - *'normalize_concat_up_down'*: eine Batch-Normalisierung wird auf die Concat-, Down- und Up-Layers folgen
  - *'renorm'*: muss ein Dict sein, das die Schlüssel *'start'* und *'end'* enthält, die die Start- und Enditerationen zur Lockerung von r_max und d_max angeben. Kann die Schlüssel **'r_max_value'** und **'d_max_value'** enthalten, um die Maximalwerte von r_max und d_max anzugeben. Diese sind standardmäßig 3 und 5. Bei nicht **False**/**None** wird die Batch-Renormalisierung anstelle der Batch-Normalisierung verwendet
  - *'groups'*: legt die Anzahl der Gruppen pro GroupNorm-Layer fest, wenn *normalization* = **'group_normalization'**. Wert 1 entspricht der Layer-Normalisierung
  - *'channels_per_group'*: alternative Möglichkeit, die Anzahl der Gruppen pro GroupNorm-Layer festzulegen: legt die Anzahl der Feature-Kanäle pro Gruppe fest. Wert 1 entspricht der Instanz-Normalisierung
- *num_conv*: Anzahl der Convolutional Layers in den Aufwärts- und Abwärtspfaden jeder Auflösungsebene (Standardwert: **2**)
- *down_conv*: bei **True** wird im Abwärtspfad anstelle von Max-Pooling ein Convolutional Layer verwendet (Standardwert: **False**)
- *up_conv*: bei **True** wird im Aufwärtspfad anstelle von Upsampling ein Deconvolutional Layer verwendet (Standardwert: **True**)
- *spade_channels*: wenn der Wert > 0, wird diese Anzahl von Kanälen zur Eingabe hinzugefügt, die eine Segmentierungs-

karte enthalten, um die Convolution im Aufwärtspfad räumlich zu denormalisieren (SPADE). Dies kann zur Verfeinerung einer bestehenden Segmentierung verwendet werden. Da SPADE eine vorherige Normalisierung denormalisiert, muss die *normalization* = **'batch_normalization'** oder **'group_normalization'** sein (Standardwert: **0**)

Darüber hinaus gibt es einige weitere einstellbare Parameter des U-Nets, die jedoch für unsere Forschung von geringer Bedeutung sind, darunter *prevent_bottleneck* (Standardwert: **False**), *residual* (Standardwert: **False**), *thin_downpath* (Standardwert: **False**), *no_bypass* (Standardwert: **False**), *input_normalization* (Standardwert: **None**), *activate_up_down_conv* (Standardwert: **False**), *keep_up_filters_constant* (Standardwert: **False**) und *_bottom_belongs_to_downpath* (Standardwert: **True**). Bei diesen Parametern kamen grundsätzlich die Standardwerte zum Einsatz.

Wichtig, um Unterschiede bewerten zu können, ist eine systematische Auswertung, wie z.B.:

- Varianten der Vorverarbeitung: unterschiedliche Voxelsizes, Resampling in zwei Dimensionen versus drei Dimensionen, andere Betrachtungsrichtung statt Transversal, SPS, usw.
- Varianten der Architektur: grundlegende U-Nets mit unterschiedlicher Ebenen-Anzahl, 3D-U-Nets, anisotrope U-Nets, unterschiedliche Aktivierungsfunktionen (ReLU, SeLU, pReLU, leaky ReLU, Swish, LiSHT, Mish, ...), usw.
- Lossfunktionen und andere Lernparameter: Cross-Entropy versus Dice, unterschiedliche Batchsizes, Gradientenmittelwertbildung, unterschiedliche Gewichtung, usw.

Basierend auf den obigen Ideen haben wir eine Reihe von Experimenten mit dem U-Net geplant. Wir wollten herausfinden, ob ein tieferes Netz und damit ein größeres rezeptives Feld bei kleinen Voxeln zu besseren Ergebnissen führt. Dazu haben wir zwischen 1,5 und 2,5 Voxelsizes variiert. Wir wollten auch ausprobieren, ob das Verwenden von 3D-Informationen und des SPS zu besseren Ergebnissen führt.

Da die 3D U-Nets in der Anzahl der Level nur klein sein konnten (Limitierung durch GPU-Speicher), haben wir die Voxelsizes größer gewählt, damit die Größe des rezeptiven Felds zu der Größe der zu segmentierenden Strukturen passt (siehe Kapitel 4.5.1).

Diese Experimente wurden zunächst für die U-Net Architektur konzipiert. Da wir uns allerdings früh im Projekt auf die Architektur des U-ResNets fokussierten und diese schließlich auch bei der KiTS21 Challenge verwendeten, wurden die oben genannten Trainingsserien stattdessen mit der U-ResNet Architektur durchgeführt (siehe Kapitel 5.3).

Weitere zunächst geplante Trainingsserien, wie der Vergleich von U-Nets mit verschiedenen Lossfunktionen und Learningraten, wurden nicht ausreichend oft durchgeführt, um an dieser Stelle Erkenntnisse darüber zu zeigen. Stattdessen wurden allerdings andere Parameter und Konfigurationen, wie die Patchsize und Data Augmentation, in Trainingsserien mit dem U-Net verwandten U-ResNet untersucht (siehe Kapitel 5.3).

4.2.2 AU-Net

AUTOR*IN: MARKUS RINK

Im Laufe des Projekts wurde das Tensorflow Backend auf Version 2 aktualisiert, wodurch eine Korrektur im Keras AU-Net nötig wurde. Außerdem enthielt die Implementierung einen Fehler, bei dem eine Variable falsch benannt wurde, wodurch die Attention Gates den Uppath ersetzten.

Das U-Net hat eine bestimmte Anzahl an sinnvollen Eingabegrößen (siehe Tabelle 3.2). Da die Attention Gates eine zusätzliche Berechnung auf der Skip Connection machen, ohne dessen Größe zu verändern, sollten die selben Eingabegrößen auch für das AU-Net gelten. Allerdings werden bei der derzeitigen Implementierung Fehler für viele dieser Eingaben geworfen. Da das Debugging für die Versionskonvertierung bereits sehr lange gedauert hat, haben wir RedLeafs `network_dump` genutzt um eine funktionierende Eingabegröße zu finden.

Die `network_dump` Textdatei zeigt alle Layer für ein instanziiertes Netzwerk tabellarisch aufgelistet. Die Größen der Layer können aber nur angezeigt werden, wenn die Eingabegröße bekannt ist. Diese können bei Convolutional Layer auch erst während der Anwendung bestimmt werden, welches bei RedLeaf der Standard ist. Der AU-Net Implementierung wurde eine zusätzliche Aufrufvariabel `input_channel` hinzugefügt, in welchem die Eingabegröße fixiert wird. Dadurch werden alle Layergrößen im `network_dump` angezeigt und eine passende Eingabegröße kann durch ausprobieren verschiedener Größen gefunden werden.

4.2.3 U-ResNet

AUTOR*IN: NIKLAS AGETHEN

Im Folgenden wird die Anwendung des in Kapitel 3.4.3 beschriebenen U-ResNets dargestellt. Diese beinhaltet die Erweiterungen der RedLeaf Bibliothek, um die für das U-ResNet Training notwendige Architektur als auch die Beschreibung der durchgeführten Trainings mit unterschiedlichen Konfigurationen dieser Architektur.

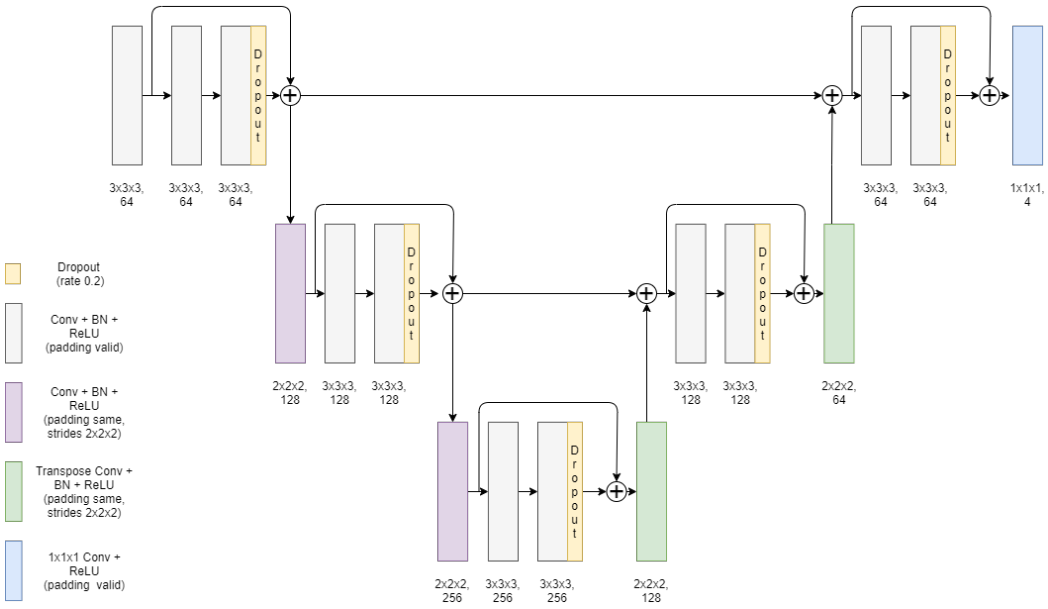


Abbildung 4.5: U-ResNet Architektur mit 3 Leveln

Zunächst soll die RedLeaf Bibliothek durch die U-ResNet Architektur erweitert werden. Zwar existiert die Architektur zu Projektbeginn bereits, allerdings wurde für die Implementierung die veraltete Tensorflow Bibliothek verwendet. Da die U-ResNet Architektur im weiteren Verlauf für die KiTS21 Challenge verwendet werden soll, verspricht das Implementieren der Architektur in der moderneren und besser angebundenen Keras Bibliothek ein vereinfachtes und effizienteres Training von U-ResNet Modellen. Für die Implementierung der Keras U-ResNet Architektur dienen somit die Tensorflow U-ResNet und die Keras U-Net Implementierung als Orientierung.

Die U-ResNet Architektur setzt sich im Wesentlichen aus Residualblöcken, Convolutionschichten mit Strides zum Downsampling und Transpose-Convolutionschichten zum Upsampling zusammen. Durch die Kombination von Downsampling im Encoder und Upsampling im Decoder ensteht die U-Net typische Struktur des Netzes (siehe Bild 4.5). Gemäß den RedLeaf-Standards erlaubt die Architektur einige konfigurierbare Einstellung, die über eine Konfigurationsdatei angepasst werden können. Dazu zählen:

- Anzahl an Leveln (Default: 5)
- Anzahl an Convolutionschichten pro Residualblock (2)
- Kernelgröße der Convolutionschichten $(3 \times 3 \times 1)$
- Paddingform (Valid Padding)
- Anzahl an Filtern in den Convolutionschichten des ersten Levels (64)
- Normalisierungsform (Batchnormalisierung)
- Dropout aktiviert / deaktiviert (deaktiviert)
- Aktivierungsfunktion (ReLU)
- Aktivierungsfunktion der letzen Schicht (ReLU)
- Softmax aktiviert / deaktiviert (aktiviert)
- Anzahl an Kanälen des Eingabebilds (1)
- Anzahl an Ausgabekanälen (2)

Durch die Angabe von Defaultwerten ist eine Anpassung in der Konfiguration nur bei Änderungen notwendig. Die getroffenen Parameter sind zudem allgemeingültig, betreffen also alle entsprechenden Layer der Architektur (z.B. Aktivierungsfunktion oder Normalisierungsform). Die Theorie hinter den einzelnen Parametern findet sich in Kapitel 3.1 wieder.

Äquivalent zur U-Net Architektur verfügt das U-ResNet über eine konfigurierbare Anzahl an Leveln. Jedes Level setzt sich aus einem Residualblock und einer weiteren Convolutionschicht zum Downsampling im Encoder oder Upsampling im Decoder zusammen. Somit legt die Anzahl der Level nicht nur die Menge an Convolutionschichten, sondern auch die minimale Bildauflösung fest. Innerhalb eines Residualblocks kommen wiederum eine konfigurierbare Anzahl an Convolutionschichten zum Einsatz (siehe Parameterliste oben). Die Kernelgröße, die Paddingform sowie die Anzahl an Filtern werden ebenso über die Parameter festgelegt, wobei sich die Anzahl an Filtern in den Leveln des Encoders jeweils verdoppelt und in den Leveln des Decoders halbiert. Jede Convolutionschicht wird durch eine optionale Normalisierungsschicht und die gewählte Aktivierungsfunktion kompletiert. Pro Residualblock wird unabhängig von der Anzahl an Convolutionschichten einmalig Dropout angewandt, falls entsprechend in den Prametern festgelegt. Abschließend wird die Shortcutverbindung zu der aktuellen Ausgabe addiert und damit das Konzept des Residuums umgesetzt (siehe Kapitel 3.4.3). Dabei muss beachtet werden, dass sich die Eingabegröße des Residuals bei der Paddingform *valid* von der Bildgröße vor der Addition unterscheidet, da die Ränder des „Bildes“ pro Convolutionschicht abhängig von der Kernelgröße abgeschnitten werden. Die Addition erwartet an der Stelle allerdings zwei gleichgroße „Bilder“, was durch das Zuschneiden des Eingabebildes um das Zentrum realisiert wurde. Dabei konnte auf vorhandene Methoden der RedLeaf Bibliothek zurückgegriffen werden.

Die Downsamplingschichten zur Reduzierung der Bildauflösung im Encoder nutzen eine weitere Convolution mit Strides zur Halbierung der Bildgröße. Dabei wird ebenfalls die dem Level entsprechende Filteranzahl verwendet, allerdings kommt unabhängig von der konfigurierten Paddingform an dieser Stelle *same* Padding zum Einsatz, um die Bildgröße genau auf die Hälfte zu reduzieren. Neben der Convolutionschicht wird wie-

derum die gewählte Normalisierung und Aktivierungsfunktion genutzt. In den Transpose-Convolutions des Upsamplings (siehe Kapitel 3.4.1) dienen ebenfalls Strides zur Verdopplung der Bildgröße. Wie beim Downsampling werden dabei, abgesehen von der Paddingform, die gewählten Parameter verwendet und durch Normalisierung und Aktivierungsfunktion ergänzt. Die Level des Encoder und Decoders des selben Rangs werden zudem, wie in der U-Net Architektur beschrieben, durch Shortcuts miteinander verbunden. Die Ausgabe des Encoderlevels (vor dem Downsampling) wird zu der Eingabe des Decoderlevels (nach dem Upsampling) addiert. Bei Padding valid muss vor der Addition wiederum das Bild des Encoders auf die passende Größe geschnitten werden.

Die Gesamtarchitektur (siehe Bild 4.5) verfügt neben den erwähnten Leveln über eine zusätzliche Convolutionschicht (mit den gewählten Filter-, Padding- und Kernelkonfigurationen sowie Normalizationschicht und Aktivierungsfunktion) zu Beginn und abschließender Convolutionschicht mit Kernelsize eins. Diese Struktur wurde von dem vorhandenen Tensorflow U-ResNet übernommen. Die finale Convolutionschicht reduziert die Filteranzahl auf die gewählte Ausgabefilteranzahl und verwendet die gewünschte Aktivierungsfunktion für die letzte Schicht des Netzwerks. Falls entsprechend konfiguriert, folgt darauf die Softmaxschicht. Durch die Verwendung der vorhandenen RedLeaf Methoden und die anpassbaren Parameter, kann die Architektur sowohl 2D als auch 3D Bilder flexibler Größen verarbeiten. Im Hinblick auf die Inferenz wurde zudem die Ausgabe der gewählten Einstellungen gemäß der Keras U-Net Implementierung realisiert. Diese ermöglicht das Speichern der für die Inferenz relevanten Einstellungen, wie Padding, minimaler Ausgabebildgröße, usw., in einer Datei des Modell-Ausgabeordners. Während der Ausführung der Inferenz kann diese dann ausgelesen und die entsprechenden Einstellung getroffen bzw. überprüft werden.

Da das U-ResNet für die Anwendung als Segmentierer konzipiert ist, setzt das Training der U-ResNet Architektur die Verwendung der von RedLeaf bereitgestellten *SegmentationTask* voraus. Diese organisiert das Training indem es u.a. die einzelnen Komponenten des Trainings, inklusive Preprocessing und Postprocessing, miteinander verknüpft, sowie die Optimierung des Modells übernimmt. Durch die Verwendung der

SegmentationTask lassen sich die U-ResNet Modelle über eine einfache Konfigurationsdatei im MEVIS Cluster ausführen. In der Konfigurationsdatei zur *SegmentationTask* werden bspw. die Batchsize, Lossfunktion, Optimizer und Lernrate definiert. Während der im weiteren Verlauf beschriebenen U-ResNet Trainings wurde stets der Dice-Loss und Adam-Optimizer mit einer Lernrate von 0,0001 verwendet. Als Grundlage für das Pre- und Postprocessing der Trainings dienten die in den Kapiteln 4.5.2 und 4.5.3 beschriebenen Methoden, an denen für die einzelnen Experimente nur Kleinigkeiten angepasst wurden (siehe unten).

Wir befassten uns zunächst mit der bereits abgelaufenen KiTS19 Challenge und starteten einige Trainingsserien mit dessen Datensatz. Für die während des Projekts angekündigte und gestartete KiTS21 Challenge übernahmen wir die gewonnen Erkenntnisse der zuvor abgeschlossenen Trainings und starteten weitere Traingsserien. Die Basiskonfiguration mit der die U-ResNets trainiert wurden, orientierte sich grob an den Standardwerten der U-(Res)Net Architektur:

- **Anzahl an Leveln**: 4
- **Anzahl an Convolutionschichten pro Residualblock**: 2
- **Kernelgröße der Convolutionschichten**: $3 \times 3 \times 1$ *(somit 2D)*
- **Paddingform**: Valid Padding
- **Anzahl an Filtern in den Convolutionschichten des ersten Levels**: 64
- **Normalisierungsform**: Batchnormalisierung
- **Dropout**: Deaktiviert
- **Aktivierungsfunktion**: ReLU
- **Aktivierungsfunktion der letzen Schicht**: ReLU
- **Softmax**: Aktiviert
- **Voxelsize** (Preprocessing): 2,5 *(isotrop)*

Der Vergleich von U-ResNet Modellen unterschiedlicher Tiefe sollte zunächst Aufschluss über die Bedeutung des rezeptiven Felds liefern. Durch das Hinzufügen oder Entfernen von Leveln und Convolutionschichten im

Residualblock verändert sich das rezeptive Feld des Modells (siehe Tabelle 3.4). Unter Berücksichtigung der Analyse in Kapitel 4.5.1 wurde die Architektur so gewählt, dass das rezeptive Feld mindestens die gesamte zu segmentierende Struktur abbilden kann. Dabei wurde sich an dem Ansatz des nnU-Nets orientiert, die Architekturentscheidungen auf Basis der Daten zu treffen. Auch die Voxelsize wurde an dieser Stelle in Betracht gezogen, da eine Veränderung der Voxelsize gleichzeitig die Größe der zu segmentierenden Strukturen in Voxeln verändert. Somit galt es an dieser Stelle die optimale Kombination aus Netztiefe und Voxelsize zu finden. Im Sinne der *Ablation Study* wurden ebenfalls Modelle mit konträren Einstellungen, d.h. kleineren rezeptiven Feldern, gestartet. Zudem stellten wir die Ergebnisse von 2D- und 3D-Modellen gegenüber. Diese Experimentenserie war ebenfalls durch das nnU-Net motiviert, da dort für die meisten Datensätze die 3D-Modelle bessere Ergebnisse erzielten [49]. Aufgrund der Hardware-Limitierungen waren wir an dieser Stelle gezwungen, neben der Umstellung auf 3D-Eingabebilder ebenfalls die Parameter Batchgröße, Patchgröße und Filtergröße anzupassen. Die bisher verwendeten Größen passten für die 3D-Bilder nicht auf die zur Verfügung stehende GPU. Auch damit haben sich die Autoren des nnU-Nets bereits befasst.

Außerdem untersuchte eine weitere Trainigsserie den Einfluss von Oversampling im Preprocessing. So wurden Trainings mit Oversampling- und ohne Oversampling-Strategie gestartet. Gerade bei der KiTS21 Challenge erschien das Verwenden von Oversampling aufgrund der sehr wenigen Zystenvoxel sehr erfolgsversprechend. Die Ergebnisse der U-ResNet Modelle und deren Auswertung werden in Kapitel 5.3 detailliert beschrieben und verglichen.

4.2.4 Deep Medic

AUTOR*IN: MARCEL PLUTAT

Im Rahmen dieses Projekts wurde die Deep Medic Architektur aus Kapitel 3.4.4 zur RedLeaf Bibliothek hinzugefügt. Für die Implementierung

wurde die *Keras* Bibliothek verwendet. Des Weiteren wurde die Referenz Implementierung von Deep Medic auf Github [48] zur Orientierung betrachtet.
Grundsätzlich wurde die Architektur gemäß der theoretischen Grundlagen aus dem Paper und den Ansätzen auf Github implementiert. Für die Implementierung in RedLeaf wurden allerdings einige Änderungen vorgenommen. Zum Einen ist die Anzahl der Convolutional Layer variabel. Die Convolutional Layer werden über eine Liste als Parameter an die Methode, die die Architektur erstellt, überreicht. Dieser Ansatz sorgt dafür, dass mit mehr Elementen in der Liste die Anzahl der Convolutional Layer erhöht wird und dass die Anzahl der Filter für jeden Layer einzeln angegeben werden kann. Analog wird die Anzahl der „*Dense*“ Layer ebenfalls als eine Liste als Argument an die Methode übergeben. Die Architektur wird iterativ über die gegebenen Listen für Convolutional und Dense Layer aufgebaut. In jeder Iteration wird ein Block bestehend aus Convolution, Batch Normalization, Dropout und Aktivierungsfunktion erstellt.
Die nächste Änderung ist die Anpassung an beliebige Eingabegrößen. Die „Standardimplementierung“ von Deep Medic erwartet Eingaben der Größen `(25,25,25)` und `(19,19,19)` für die normale respektive niedrige Auflösung. Diese Eingabegrößen können im Keras Input-Layer fest hinterlegt werden, dann können allerdings nur diese Eingabegrößen verwendet werden. Da auch größere Eingaben sinnvoll sind, wurden die Eingabegrößen im Input-Layer bewusst als `(None,None,None)` angegeben. Das Netz akzeptiert dadurch beliebige Eingaben und entspricht den RedLeaf Konventionen, die die flexible Konfiguration der Architekturen vorsieht.
Die Verwendung von Dropout ist eine weitere Addition, die weder im Paper noch in der Referenz Implementierung vorhanden war. Zum Einen ist die Erweiterung um Dropout sinnvoll, da durch Dropout Overfitting verhindert werden kann. Zum Anderen wurde Dropout gewählt, da andere Netzarchitekturen in RedLeaf ebenfalls Dropout als Parameter enthalten und Deep Medic dadurch wieder den RedLeaf Konventionen entspricht.
Die Kernelgröße ist ein weiterer variabler Parameter. Vorgesehen sind für Deep Medic nur $3 \times 3 \times 3$ Kernel. In der RedLeaf Implementierung sind auch größere Kernel erlaubt, falls diese valide sind. Valide Kernel sind alle, die uniform sind, zum Beispiel $5 \times 5 \times 5$, und eine ungerade Größe

haben. Das heißt, dass gerade Kernel, wie $4 \times 4 \times 4$, nicht erlaubt sind.

Der zweite Teil der Deep Medic Implementierung hat sich mit der Erweiterung von Deep Medic um Residual Verbindungen befasst. Die Grundlage hierfür waren wieder die Theorie aus Kapitel 3.4.4 und das Github Repository von Deep Medic.
Zur Verwendung von Residual Verbindungen in der Architektur wurde der Parameter *residual_connections* eingeführt. Dieser Parameter ist ein Boolean Wert, der angibt, ob die Architektur Residual Verbindungen enthalten soll oder nicht. Die Residual Verbindungen werden in einer Liste als Parameter der Methode definiert. Die Elemente dieser Liste sind dabei die Indices, an denen eine Residual Verbindung erstellt werden soll. Um die Residual Verbindungen zu erstellen, ist ein zweiter Parameter namens *offset* angegeben. Die Residual Verbindung für einen Index i besteht letztendlich aus der Addition der Layer $i -$ offset und i. Der Parameter *offset* bestimmt folglich wie viele Convolution Blöcke in der Residual Verbindung übersprungen werden sollen und die Liste der Indices gibt die Anzahl der Residual Verbindungen an.
Falls Residual Verbindungen mit Deep Medic verwendet werden, ändert sich auch die Reihenfolge im Convolution Block auf Batch Normalization, Aktivierungsfunktion, Convolution und abschließend Dropout. Dazu kommt eine Residual Verbindung falls der aktuelle Index in der Residual Connections Liste enthalten ist.

Um Deep Medic zu verwenden, müssen zunächst die Eingaben auf den beiden Auflösungen erstellt werden. Deep Medic setzt die Verwendung vom *MultiResolutionSegmentationTask*, einem RedLeaf Preprocessing Modul, voraus um mehrere Eingaben in verschiedenen Auflösungen zu erstellen.

Zur Validierung der Implementierung von Deep Medic wurden einige Trainings gestartet, um die jeweiligen Änderungen zu überprüfen. Gemeinsamkeiten aller Trainings sind die Verwendung des *Stratified Patch Samp-*

ler im Preprocessing und Batch Normalization sowie Dropout in der Architektur.

4.3 Challengr

AUTOR*IN: ROBERT BOHNSACK

Ein großer Teil des Projektes war die Arbeit an Challengr. Dieses wurde im Verlauf des Projektes angepasst und erweitert, während die Ideen dafür größtenteils aus der Nutzung der Software entstanden, und teilweise speziell für besondere Anwendungsfälle aus dem Projekt implementiert wurden.

4.3.1 Verbesserungen

Verbesserungen von bestehenden Funktionen wurden an folgenden Stellen umgesetzt:

4.3.1.1 Browse Sessions

- Die HTML Tabelle wurde mit einer Q-Table aus dem Quasar Framework ersetzt. Diese bietet einige Verbesserungen
  - Einen Ladeindikator, falls das Backend nicht schnell genug antwortet
  - Eine Einteilung der Liste in mehrere Seiten
  - Eine Suchfunktion nach allen gelisteten Parametern
  - Eine (wenn gewollt selbst implementierte) Sortierbarkeit aller Spalten
  - Bessere Benutzerfreundlichkeit durch neues Layout
  - Leichtere Wartbarkeit und Erweiterbarkeit im Code

- Die Namen der Sessions sind nun über Challengr anhand eines Textfeldes, welches beim anklicken der Namen erscheint, anpassbar
- Die Beschreibung der Sessions, welche vorher nur in Compare Sessions verfügbar war, ist nun auch in Browse Sessions anzeigbar

4.3.1.2 Compare Sessions

- Die Untergruppen Global Results, Session Settings und Statistical Significance sind nun einklappbar
- Werte, welche in beiden Sessions gleich sind, können nun ausgeblendet werden
- Name und Beschreibung der Sessions können nun angepasst werden
- Ein Bug wurde gefixed, bei dem die Ergebnisse von Sessions mit verschiedenen Evaluationsmaßen in den falschen Zeilen angezeigt wurden
- Das Symbol „N/A“ wenn ein Wert nicht verfügbar war wurde mit „-“ ersetzt

4.3.1.3 CaseDataTable

(a) Aussehen vorher (b) Aussehen nachher

Abbildung 4.6: Verbesserungen an der CaseDataTable

Bei der CaseDataTable wurden größtenteils neue Designentscheidungen getroffen, welche die Lesbarkeit der Tabelle deutlich verbessert haben (siehe Abb. 4.6). Dazu wird die Änderung der Werte nicht mehr durch Pfeile auf beiden Seiten angegeben, sondern durch eine eigene Spalte, in welcher der Deltawert steht und farblich markiert wird, je nachdem welche Session für den jeweiligen Fall das bessere Ergebnis beinhaltet. Generell wurde die Tabelle kompakter und besser unterteilt.

4.3.1.4 Generelle Verbesserungen

Es wird nun der Zustand der ausklappbaren Elemente, wie Browse oder Compare Sessions, im Cache gespeichert, sodass diese beim neu laden nicht wieder vom Nutzer ausgeklappt werden müssen. Dazu werden die ausgewählten Sessions gespeichert, damit direkt Ergebnisse in Compare Sessions und der CaseDataTable angezeigt werden.
Außerdem wurde ein generelles Refactoring der Software durchgeführt,

welches insbesondere häufig genutzte Methoden und Styles aus Vue Komponenten in eigene Dateien auslagert.

4.3.2 Erweiterungen

Im Laufe des Projektes sind außerdem einige fehlende Funktionen aufgefallen, welche von uns ergänzt wurden.

4.3.2.1 Compare Multiple Sessions Tab

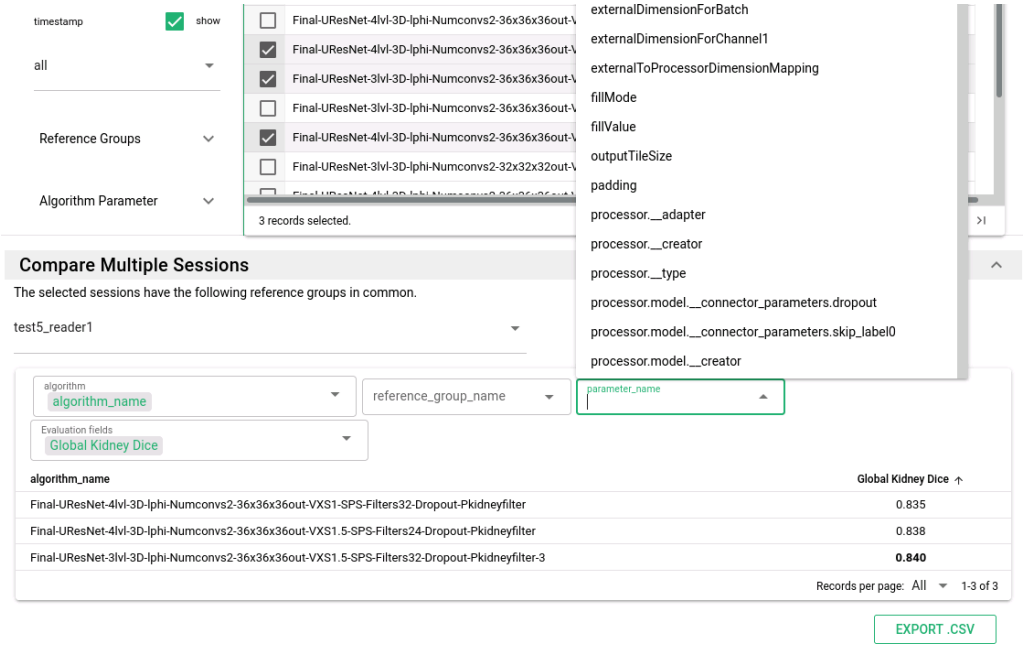


Abbildung 4.7: Neuer Tab mit Multiselect und angepasster Vergleichstabelle

Eine dieser Funktionen war der Vergleich von mehr als zwei Sessions. In Challengr war es bereits möglich jeden Parameter von zwei Sessions gegenüberzustellen und sich Unterschiede anzugucken. Allerdings musste für den Vergleich von drei oder mehr Sessions mehrmals die Auswahl geändert werden, was umständlich war.
Um diesen Anwendungsfall zu erleichtern, wurde ein neuer Tab (siehe Abb. 4.7) in der Challengr Oberfläche eingeführt. Dieser verhält sich wie

der normale Tab zum Vergleich von Sessions, nur dass in der Sessionauswahl die Selektion von beliebig vielen Sessions möglich ist. Die Werte einer Session auf die eine Seite und die der anderen auf die andere Seite zum Vergleich zu stellen war so allerdings auch nicht mehr möglich.
Deshalb wurde eine Tabelle eingeführt, in welcher alle ausgewählten Sessions als Zeilen aufgelistet wurden. Die Spalten konnten dann mit von einem Nutzer gewählten Parametern der Sessions befüllt werden. Außerdem waren die Spalten sortierbar und in den Spalten zu Evaluationsmaßen wurde der beste Wert hervorgehoben. Die Auswahl der Spalten wurde gespeichert, sodass diese bei der nächsten Nutzung erhalten bleiben. Wollte ein Nutzer die Ergebnisse erweitert auswerten oder nutzen, so gab es die Option die Tabelle als CSV Datei zu exportieren.

4.3.2.2 Evaluations Metrik Diagramm

AUTOR*IN: MARKUS RINK

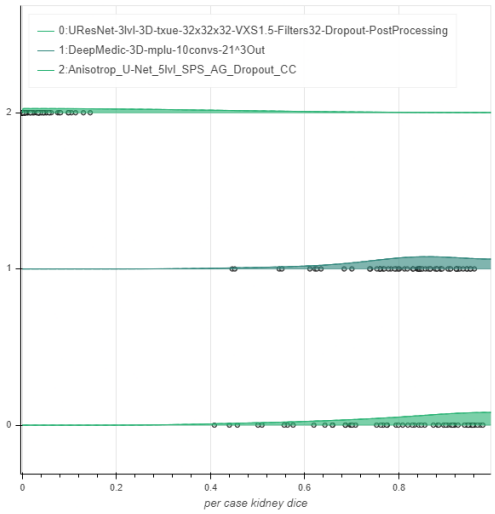


Abbildung 4.8: Evaluations Metrik Diagramm in Challengr

In Challengr können beliebige Metriken angezeigt werden. Diese liefern für jeden Fall einen Wert. Für jede Metrik wird der Wertebereich und das Optimum gespeichert.

Das Backend bietet eine Chart API, welche Bokeh Diagramme erstellt. Bokeh ist Python basiert und generiert ein interaktives Diagramm in HTML und Javascript.

Das Diagramm wird für eine ausgewählte Metrik berechnet. Für jedes ausgewählte Netz zeigt es einen Scatterplot von allen Fällen. Um die Verteilungen besser zu vergleichen, wird eine Kerndichteschätzung eingezeichnet. Dabei wird ein Kern, welcher z.B. eine Normalverteilung bildet, auf jeden Datenpunkt gelegt und diese Kurven zusammen addiert. Diese Abschätzung ist optimal für eine Normalverteilung mit einer Mode. Die den Daten zugrunde liegende Verteilung ist zwar viel komplexer, um aber einen visuellen Eindruck zu bekommen, ist diese Methode ausreichend.

Diese Implementierung berücksichtigt Intervalle mit und ohne Begrenzung. Bei einem unbegrenzten Intervall wird die Kurve 10% über dem Minimum und Maximum Wert hinaus gezeichnet, um auch niedrige Auftrittswahrscheinlichkeiten in diesem Bereich noch anzuzeigen. Bei geschlossenen Intervallen muss der überschüssige Bereich auf die restliche Kurve addiert werden. Dafür wird die Kurve an der Intervallgrenze gespiegelt, aufeinander addiert und abgeschnitten. Dadurch erhält man wieder die selbe Fläche unter der Kurve und somit eine bessere Abschätzung der Wahrscheinlichkeit innerhalb des Intervalls.

Eine Alternative, aber nicht verwendete Methode, wäre ein Ignorieren der Kerndichteabschätzung über der Intervallgrenze und ein entsprechendes Skalieren um die entfernte Kurvenfläche. Keine der beiden Varianten behebt die falsche Annahme der Abschätzungsmethode, dass es keine Grenzen gäbe. Dadurch ist keine der beiden Abschätzungen der anderen klar vorzuziehen.

Beim Hovern über die Punkte wird die zugehörige Fallnummer sowie erneut das evaluierte Netz und dessen Challengr UID angezeigt. Da die Legende das Diagramm teilweise verdeckt, lässt es sich per Doppelklick ein- und ausblenden. Außerdem können die Namen der Modelle beliebig lang werden. Um die y-Achsen Beschriftung möglichst kompakt zu gestalten, werden die Modellnamen nur in der Legende, bzw. dem Hover-Text angezeigt.

4.4 Docker

AUTOR*IN: TIMO GÜNNEMANN

Wir haben jeweils für Challengr und RedLeaf unser eigenes Docker-Skript und eine gitlab.ci geschrieben. Jedes Mal, wenn auf den Deep Anatomy Master von Challengr und RedLeaf gepusht wird, wird automatisch ein neues Image gebaut und in die Registry von unserem Projekt für RedLeaf und Challengr gepushed. Die Änderungen können dadurch auf dem Cluster genutzt werden, ohne dass diese erst auf den offiziellen Master von MEVIS committed werden müssen. Damit das Webinterface von Challengr nicht immer lokal gestartet werden muss, läuft auf Nomad eine Challengr-Instanz. Diese wird nach dem Bau eines Images über die CI automatisch neu gestartet, sodass die Änderungen gleich auf dem Cluster verfügbar sind. Zur Nutzung der selbst implementierten Architekturen zum Training auf dem QuantMed-Dashboard ist es möglich, das Docker-Skript manuell auszuführen und somit eigene Images zu erstellen. Dadurch kann man die Architektur von den Feature-Branches testen, ohne diese auf den (projekteigenen) Master zu ziehen.

4.5 KiTS21-Challenge

4.5.1 Datensatz

AUTOR*IN: NIKLAS AGETHEN

Für die KiTS Challenge stellen die Organisationen einen Datensatz von 300 CT-Bildern mit Annotation zur Verfügung. Gemäß den erfolgsversprechenden Erkenntnissen des nnU-Nets sollten die Konfigurationsentscheidungen der KiTS Trainingsserien auf Basis der Eigenschaften des Datensatzen getroffen werden. Daher wurde vorab eine detaillierte Analyse der Daten vorgenommen.

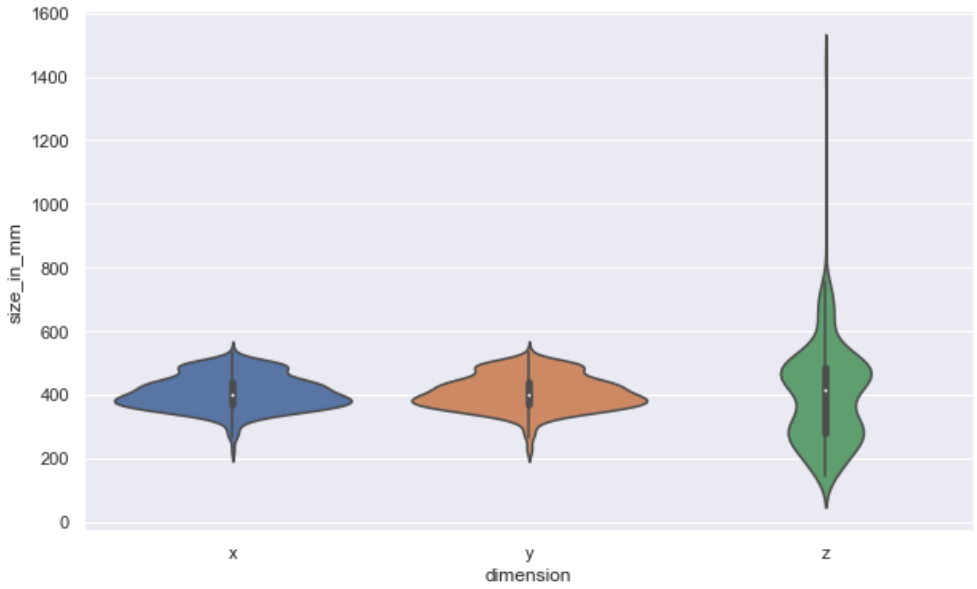


Abbildung 4.9: Verteilung der Bildgrößen je Dimension

Zunächst lag der Fokus auf den Bild- und Voxelgrößen des Datensatzes. Mittels Python-Code konnten diese für den gesamten Datensatz berechnet und visualisiert werden. Für die Berechnungen der Bildgröße wurden die Bilder zunächst auf die Voxelsize von $1mm$ und die Bildorientierung auf transversal angepasst. Das Ergebnis der Bildgrößen zeigt, dass diese in x- und y-Richtung weniger stark voneinander abweichen als in z-Richtung (siehe Abbildung 4.9). In z-Richtung beträgt die minimale Größe $142mm$, die maximale Größe $1437mm$ und der Durchschnitt etwa $398mm$. In x- und y-Richtung liegt die minimale Größe jeweils bei $224mm$, die maximale Größe bei $533mm$ und der Durchschnitt in beiden Dimensionen bei etwa $407mm$. Die Bildgröße wurde bei den Architektur-Parametern bspw. bei der Patchsize in Betracht gezogen und soll eine Orientierung dafür liefern, welche maximale Patchsize sinnvoll erscheint. Dabei muss selbstverständlich die vom jeweiligen Modell angewandte Voxelsize berücksichtigt werden.

Für die Voxelsizes des KiTS Datensatzes zeichnet sich eine ähnliche Beobachtung wie bei den Bildgrößen ab: In z-Richtung besitzen die Bilder einen deutlich größeren Wertebereich als in x- und y-Richtung (siehe Abbildung 4.10). So existieren in z-Richtung sehr feine Voxelsizes von $0,5mm$ als auch sehr grobe mit etwa 5mm. In x- und y- hingegen lie-

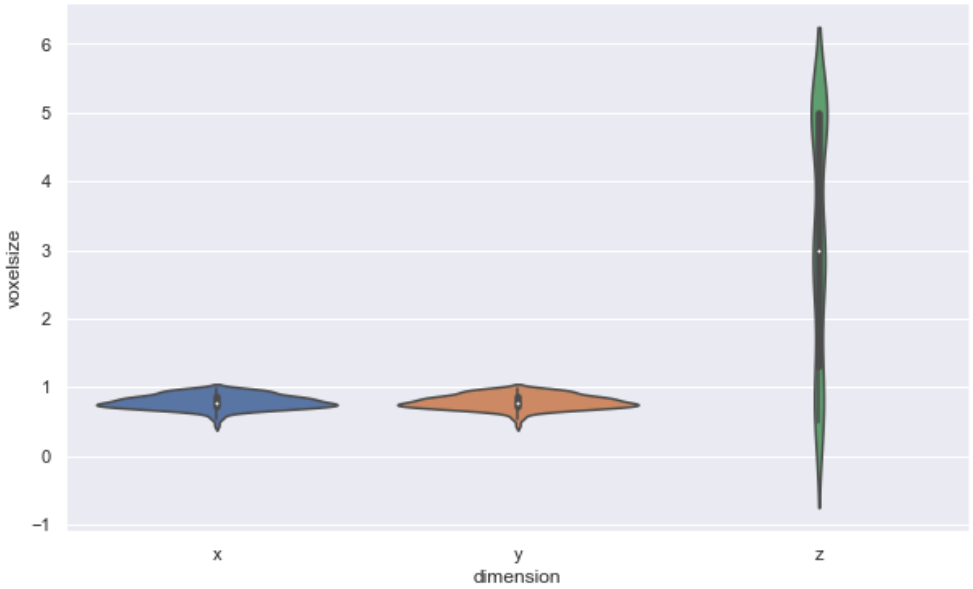


Abbildung 4.10: Verteilung der Voxelsizes je Dimension

gen die Voxelsizes zwischen $0,44mm$ und etwa $1mm$. Die Verteilung der unterschiedlichen Voxelsizes ist bspw. bei der Auswahl der Voxelsize im Preprocessing der Modelle relevant.

Als ein weiterer Schritt der Datensatzanalyse wurden die Größen der zu segmentierenden Strukturen, bestehend aus Niere, Tumor und Zyste, betrachtet. Diese kann u.a. bei der Einschätzung des rezeptiven Felds einer Architektur hinzugezogen werden. Mittels einer *Connected Component* Analyse wurden dazu die Strukturen in den einzelnen Bildern separiert, auf die Voxelsize von 1mm angepasst und deren Größe berechnet. Die *Connected Component* Analyse separiert in einigen Fällen einzelne Läsionen in viele kleine, was zu einer verzerrten Darstellung der Läsionsgrößen führt. Daher wurde bei der Berechnung der Strukturgrößen zusätzlich ein Threshold eingeführt, der Läsionen kleiner als 6mm in jeder Dimension ausschließt. Der Grenzwert wurde dabei so gewählt, dass möglichst die wahren, kleinen Läsionen erhalten bleiben, und möglichst viele der fehlerhaften kleinen Zysten aussortiert werden. Dennoch werden durch diese Umsetzung weiterhin nicht alle fehlerhafterweise segmentierten Strukturen entfernt (wenn größer als 6mm in einer Dimension), die Größen der berücksichtigten Strukturen werden durch die fehlerhaft getrennten Läsionen als kleiner erachtet und einige tatsächliche Lä-

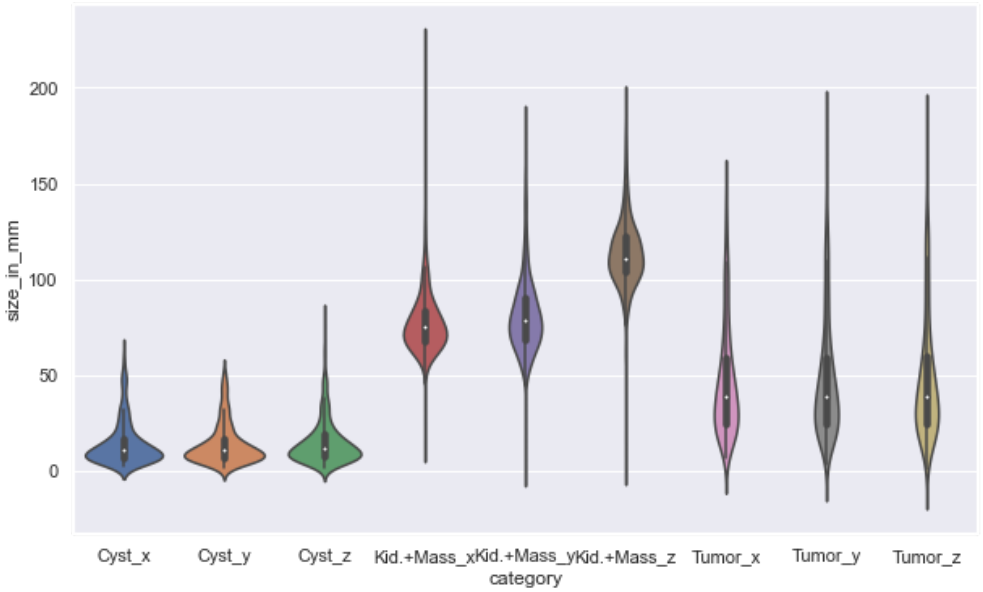


Abbildung 4.11: Verteilung der Strukturgrößen je Dimension

sionen werden ignoriert (wenn kleiner als $6mm$ in jeder Dimension). Dies muss bei der Betrachtung der Abbildung 4.11 und bei der Eindordnung der Größen berücksichtigt werden.

Die Abbildung zeigt die Größen der Kidney-and-Masses, des Tumors und der Zyste für die Dimensionen getrennt an. Der Durschnitt der Kidney-and-Masses, d.h. der zusammenhängenden Struktur aus Niere, Tumor und Zyste, ist in x- und y-Richtung durchschnittlich etwa $80mm$ groß, in z-Richtung etwa $115mm$. Hervorzuheben ist hier ebenfalls die maximale Größe von $221mm$ in x-, $179mm$ in y- und $190mm$ in der z-Dimension, die sich recht deutlich vom Durchschnitt unterscheidet und belegt, dass sich die Labelgrößen teilweise recht stark voneinander abweichen. Auch die durchschnittlich kleineren Zystengrößen gegenüber den Tumoren können aus der Abbildung entnommen werden. Die Zysten weisen eine Durchschnittsgröße von ca. $15mm \times 15mm \times 15mm$ auf, die Tumore hingegen etwa $48mm \times 48mm \times 48mm$. Die Durchschnittsgrößen wurden bspw. bei der Auswahl der Voxelsize in den KiTS Trainingsserien berücksichtigt.

4.5.2 Preprocessing

AUTOR*IN: MARKUS RINK

In der Challenge wird ein nnU-Net als Baseline bereitgestellt.

Alle CT Grauwerte wurden auf -1000HU nach unten begrenzt. Wir haben uns auf einen Data Augmentation Schritt geeinigt, welcher die X-Achse spiegelt, also die Nieren Position in transversaler Ansicht tauscht, Skalierung von $\pm 10\%$ und Rotation benutzt. Der Rotationswert wird zufällig von einer Normalverteilung gezogen, wobei $15°$ auf der ersten Standardabweichung liegt. Dadurch sollen die Variationen von Organen im Körper und Positionen der Menschen im CT-Scanner nachgeahmt werden.

Wir haben mit Voxelgrößen von $0,8mm$ bis $2,5mm$ experimentiert. Anstelle von gewichteten Eingaben benutzen wir Oversampling. Beim Analysieren des KiTS21 Datensatzes fällt auf, dass es sehr wenig Zysten Voxel gibt (siehe Abschnitt 4.5.1). Mit dem Oversampling generieren wir Batches, bei denen 50% mindestens einen Voxel Tumor oder Zyste, 25% mindestens eine Voxel Zyste und 25% beliebige Voxel enthalten. Wegen der Patchgröße ist die Anzahl an Zysten Voxeln pro Batch trotzdem noch unter $0,5\%$.

Wir teilen den Datensatz in kleinere Patches, da die verfügbare Hardware nicht die volle Bildgröße unterstützt und gleichmäßige Batchgrößen den Speicher auch besser ausnutzen können. Das Verhältnis Patch- und Batchgröße bildet einen Kompromiss von statistischer Unabhängigkeit und effizienter Datennutzung. Ein Patch enthält Datenpunkte eines einzelnen Patienten, wodurch die Voxel statistisch abhängiger sind, als würden sie von verschiedenen Patienten kommen. Dagegen ist ein großer Patch gegenüber mehreren kleinen Patches effizienter, weil bei gleicher Ausgabegröße weniger Paddingvoxel benötigt werden.

Durch das valid Padding und überlappenden Patches entstehen keine Ränder in einer zusammengesetzten Ausgabe. Die Padding-Größe ist abhängig von der jeweils trainierten Architektur, speziell der Anzahl an Rechenoperationen.

4.5.3 Postprocessing

AUTOR*IN: LENA PHILIPP

Bei der Betrachtung der Ergebnisse der Inferenz fallen Schwächen bei der Segmentierung auf, die teilweise durch nachträgliche Korrekturen gelöst werden können. Die folgenden Ungenauigkeiten treten bei allen unseren Modellen, die auf den Daten der KiTS21-Challenge trainiert wurden, auf ähnliche Weise auf. Zu diesen Problemen gehören falsch-positive Artefakte, die nicht mit den Nieren verbunden sind und ungenau segmentierte Randregionen der Läsionen. Die folgenden Ansätze sollen dazu dienen, dies zu korrigieren.

4.5.3.1 Connected Components einfach

Beim ersten Ansatz wurden die beiden größten verbundenen Strukturen über das RedLeaf Modul *ConnectedComponents* bei der Inferenz bestimmt. Die Grundidee davon ist, dass in den meisten Fällen Bilder mit zwei Nieren betrachtet werden und die zwei größten verbundenen Strukturen daher die Nieren sein sollten. Durch diesen Ansatz sollen falsch-positive, nicht mit den Nieren verbundene Regionen entfernt werden. Diese neue Maske wurde dem Originalbild hinzugefügt und gespeichert.

4.5.3.2 Connected Components Niere

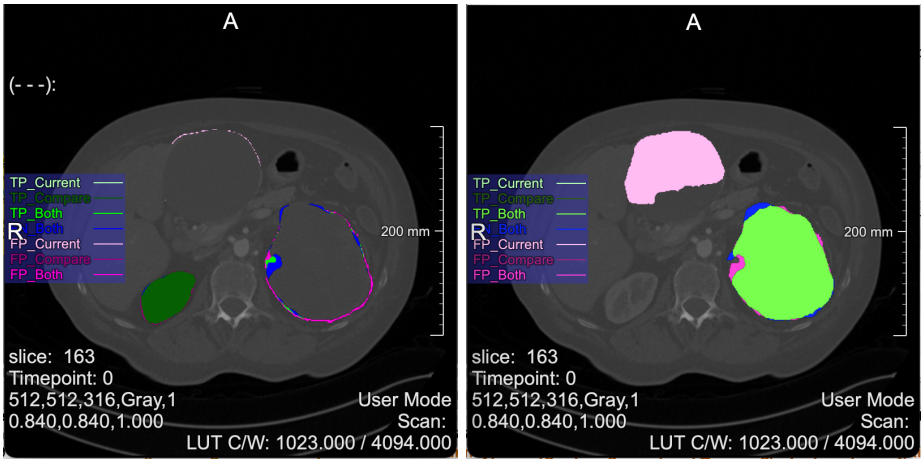


Abbildung 4.12: Fall 102, Current: ohne Vorverarbeitung der Niere, Compare: mit Vorverarbeitung der Niere. Links: Segmentierung Niere. Rechts: Segmentierung Tumor

Ein Problem des vorhergehenden ersten Ansatzes ist, dass nicht nur die Artefakte entfernt werden, sondern teilweise auch Nieren. Stattdessen werden größere falsch-positive Tumorstrukturen ausgewählt, wie Abbildung 4.12 zeigt. In diesem Fall wurde die Leber fälschlicherweise als Tumor segmentiert. Das Volumen übersteigt das der Nieren, wodurch nur eine Niere bei der Nachverarbeitung ausgewählt wurde. Um dies zu beheben wurden im zweiten Ansatz zunächst nur die Nieren betrachtet, siehe Abbildung 4.13 B. Nun wurden die verbundenen Komponenten bestimmt, die ein Volumen über 20ml besitzen. Aus den Komponenten, die übrig bleiben, wurden die zwei größten ausgewählt.

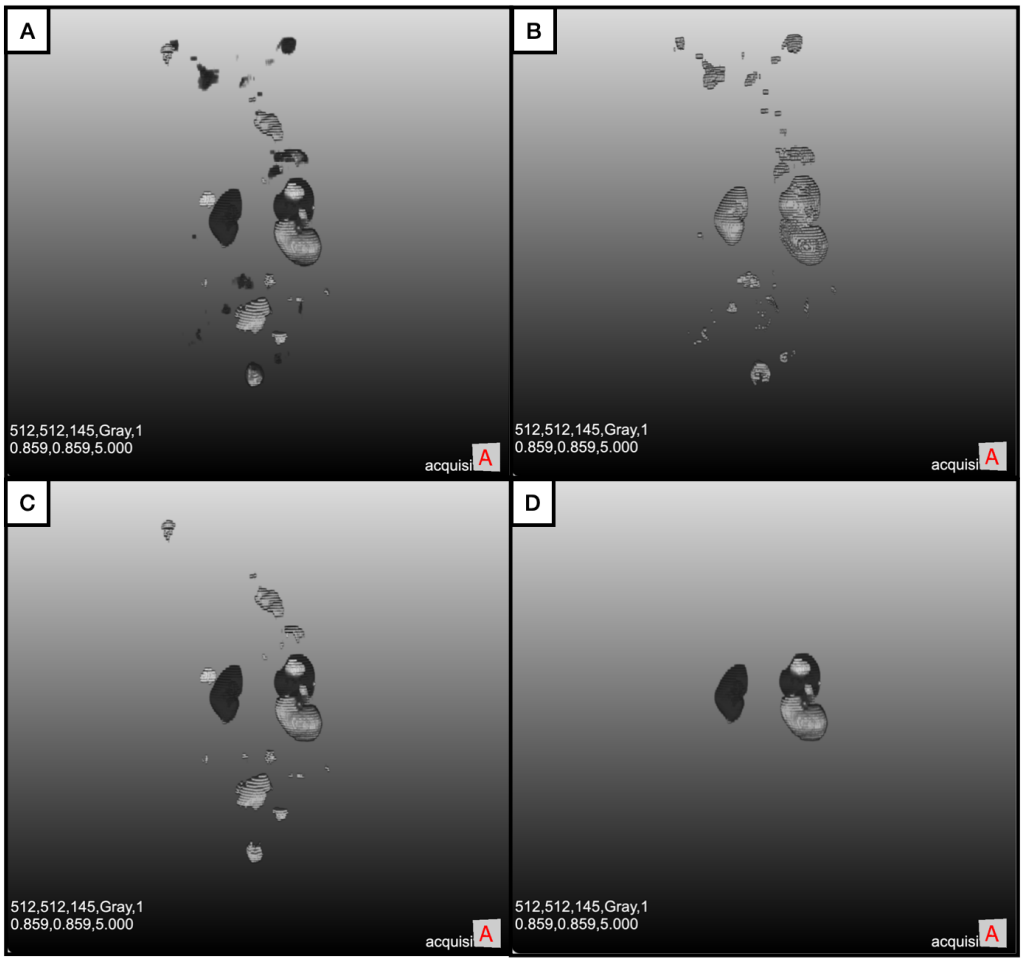


Abbildung 4.13: Case 73. A: ohne Postprocessing. B: Segmentierung der Niere vor dem 1. Nachverarbeitungsschritt. C: Verbindung der ausgewählten Nierenregionen und der Segmentierung von Tumor / Zyste. D: Komponenten mit Nierenanteil

Im nächsten Schritt wurden die als Tumor oder Zyste segmentierten Regionen mit den im vorherigen Schritt ausgewählten Nieren wieder zusammen geführt, wie Abbildung 4.13 C zeigt. Es wurden erneut die verbundenen Komponenten bestimmt und die Strukturen ausgewählt, die einen Nierenanteil besitzen.

4.5.3.3 MajorityVote Läsion

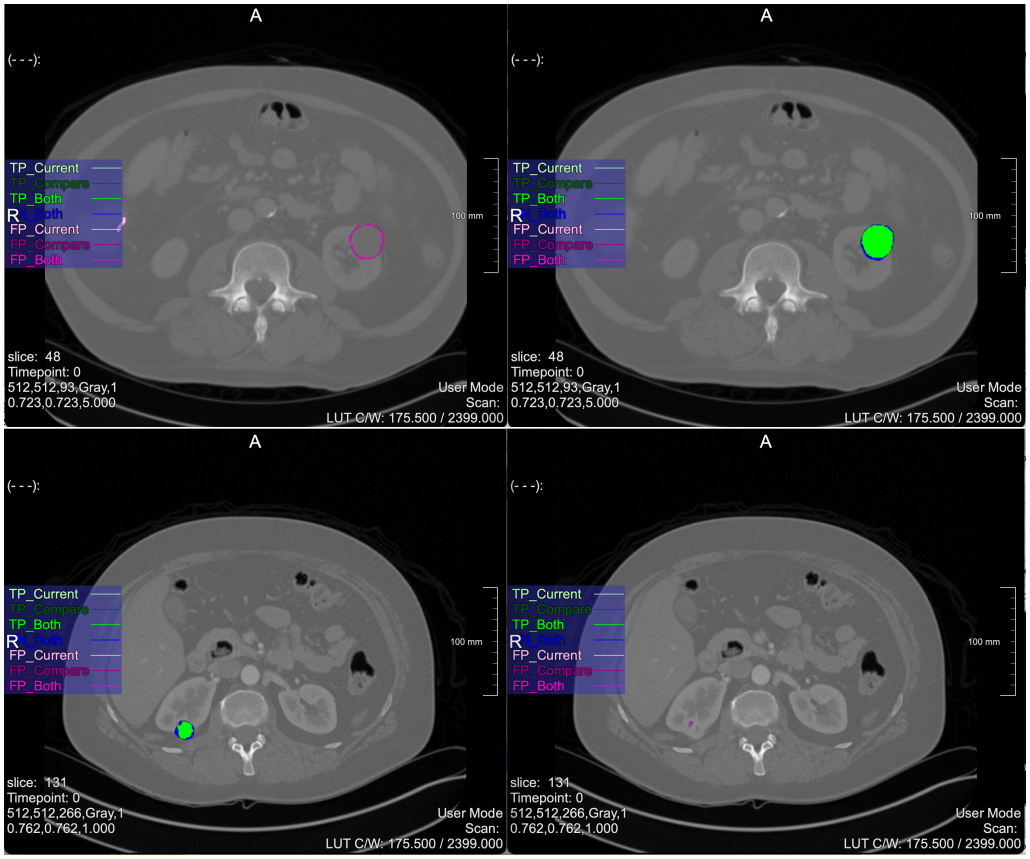


Abbildung 4.14: Oben: Case 270. Links: falsch-positiv segmentierter Tumor. Rechts: falsch-negativ segmentierte Zyste. Unten: Case 229. Links: segmentierter Tumor mit falsch-negativen Anteilen. Rechts: falsch-positiv segmentierte Zyste.

Eine weitere Auffälligkeit betrifft die Unterscheidung von Zysten und Tumoren. Ränder von Tumoren werden fälschlicherweise als Zysten segmentiert und umgekehrt, wie die Abbildungen 4.14 zeigen. Um dieses Problem zu lösen wurde das bisher beschriebene Postprocessing erweitert. Dafür wurden die Tumore und Zysten ohne die Niere betrachtet und die jeweiligen Anteile in einer Komponente bestimmt. Nun wurden die Anteile der Klasse, die in der betrachteten Komponente unterliegt, ersetzt durch die dominantere Klasse (d.h. die Klasse mit den meisten Voxeln in der Komponente).

4.5.3.4 Postprocessing mit Kaskade

AUTOR*IN: NIKLAS AGETHEN

Im Folgenden wird ein alternativer Ansatz zu dem vorgestellten MajorityVote präsentiert, der ebenfalls die Korrektur der Läsionsränder beabsichtigt. Dabei kommt eine Kaskade bestehend aus einem Segmentierer und einem Klassifikator, wie in Kapitel 3.3 beschrieben, zum Einsatz. Ein Klassifikator wurde auf den segmentierten Läsionen des Segmentierer-Modells trainiert und soll lernen diese korrekt zu klassifizieren, um anschließend die Ausgabeklassen der segmentierten Läsionen des Segmentierers zu korrigieren. In diesem konkreten Anwendungsfall, soll der Klassifikator die Klasse einer zusammenhängenden Läsion in der Segmentiererausgabe bestimmen, um so die angesprochenen Läsionsränder und das Innere einer Läsion einheitlich zu klassifizieren.

Die Umsetzung dieses Konzepts lässt sich grob in die folgenden Aufgaben unterteilen:

- **Datenvorbereitung:** Implementierung eines Modells zum Bereitstellen der Segmentiererausgaben für das Training des Klassifikators
- **Klassifikatortraining:** Implementierung der Klassifikatorarchitektur sowie Konfiguration des Trainings und Modells
- **Inferenz:** Implementierung eines Modells zur Auswertung des Klassifikators sowie Integration des Klassifikators in die Segmentierer-Inferenz

Im ersten Schritt galt es somit, die Daten für das Training des Klassifikators zu erstellen. Der Klassifikator nutzt die Ausgabebilder des jeweiligen Segmentierers und wird somit für jeden Segmentierer separat trainiert. Da sich die Ausgaben der verschiedenen Segmentierer teilweise recht deutlich unterscheiden, erschien das separierte Training des Klassifikators an dieser Stelle erfolgversprechender. Für das Erstellen der Trainingsdaten wurde der Segmentierer auf dem KiTS21 Test- und Validation-Datensatz ausgeführt, um die vom Segmentierer berechneten

Masken zu erhalten. Da der Klassifikator lediglich die Läsionen korrigieren soll, wurden anschließend einzelne, d.h. zusammenhängende Nierentumore und Nierenzysten mittels Grenzwert und Connected Component Analyse herausgefiltert. Nun lag jede vom Segmentierer erkannte Läsion als einzelnes Trainingsbild vor. Die für das Training des Klassifikators benötigten Klassenlabels der Läsionen wurden durch die jeweils häufigste Klasse innerhalb dieser Region der ursprünglichen Annotation bestimmt. Während der Datenvorbereitung wurde das Ausgabebild des Segmentierer zusätzlich um das Originalbild und das euklidische Distanzbild zu einem drei-kanaligen Bild erweitert. Das Distanzbild gibt dabei die Entfernung des jeweiligen Voxels zum Vordergrund (Nieren-, Tumor- oder Zystenvoxel) an und soll dem Klassifikator den Fokus auf die Nierenregion erleichtern.

Für das Training des Klassifikators steht in RedLeaf eine Architektur bereit, die für die Kaskade mit dem Segmentierer lediglich ein wenig erweitert werden musste. Die vorhandene Architektur verbindet Convolution- und Pooling Schichten, um die Eingabebilder zu verarbeiten und die Klassifikationsergebnisse der einzelnen Klassen in Form von One-Hot-Encoding ausgibt. Über eine Global Pooling Schicht wird die Ausgabe auf die gewünschten Ausgabekanäle reduziert. Wie in RedLeaf üblich lässt sich der Klassifikator über einige Parameter, wie Anzahl an Schichten, Paddingform, Aktivierungsfunktion, usw. konfigurieren. Die Architektur wurde zudem um zusätzliche Anpassungsmöglichkeiten erweitert, die für die Experimentenserie innerhalb der KiTS21-Challenge notwendig oder sinnvoll erschienen. Dazu gehören eine variable Anzahl an Ausgabekanälen, die optionale Softmax-Schicht als letzte Schicht der Architektur, der Paddingmodus sowie die Anzahl an Kanälen der Convolution-Schichten.

RedLeaf stellt neben der Architektur auch Code für das Training eines Klassifikators bereit: Die *ClassificationTask* verwaltet u.a. die Trainingsdaten und übernimmt die Optimierung des Modells. Wie beim Training des Segmentieres, lassen sich die Einstellungen des Trainings über eine Konfigurationsdatei festlegen und ermöglichen das Trainieren eines Klassifikators auf dem MEVIS Cluster. In der Konfigurationsdatei wurde für die Optimierung des Klassifikators das Categorical-Cross-Entropy-Loss

mit Adam Optimizer und einer Learning Rate von 0,0001 festgelegt.

Die Trainingsdaten wurden über ein Preprocessing-MeVisLab-Netzwerk vorbereitet und für das Training zur Verfügung gestellt. Im Gegensatz zum Segmentierer wurde die Overlap-Tile-Strategy (siehe Kapitel 3.4.1) beim Training des Klassifikators nicht verwendet, da pro Eingabebild eine einzelne Klassifikation gewünscht war. Um dennoch eine feste Eingabebildgröße und somit Batches mit mehr als einem Bild zu ermöglichen, wurden die Eingabebilder auf eine feste Größe erweitert oder verkleinert, indem ein Ausschnitt der gewünschten Größe um das Bildzentrum ausgeschnitten wurde.

Das Training des Klassifikators verwendet die zuvor beschriebene RedLeaf Architektur und besitzt als drei Eingabekanäle das Original- und Distanzbild sowie das Segmentierer-Label. Die vier Ausgabekanäle stehen für die vier möglichen Klassen Hintergrund, Niere, Tumor und Zyste. Zudem wurde die Softmax-Schicht bei allen Trainings genutzt und ein Klassifikator mit drei Leveln konfiguriert, da das rezeptive Feld der drei Level Architektur mit der Größe von 84 grob in der Mitte der Größen der verschiedenen Läsionen liegt (siehe Kapitel 4.5.1). Die Art des Poolings wurde auf Max-Pooling festgelegt. Um einen geeigneten Klassifikator zu finden, der die Läsionen möglichst gut klassifiziert, wurde eine Experimentenserie gestartet, die Modelle mit unterschiedlichen Konfigurationen beinhaltet. Diese variieren in der Paddingform, der verwendeten Regularisierung und der Anzahl an Kanälen in den Faltungsschichten. Im Preprocessing wurden dabei unterschiedliche Formen der Data-Augmentation angewandt. Die Details und Evaluation der Experimente finden sich im Kapitel 5.4 wieder.

Die Modelle der geplanten Klassifikatortrainings sollten zunächst in einer separaten Inferenz evaluiert werden, um den besten Klassifikator für den jeweiligen Segmentierer zu finden. Aufgrund der KiTS21 Abgabefrist und dem nahenden Projektabschluss wurde allerdings auf diese separate Inferenz verzichtet und stattdessen der Klassifikator direkt in die Inferenz des Segmentierers integriert. Eine Einschätzung des Klassifikators während und nach dem Training war über den jeweiligen Lossplot und die Confusion-Matrix des Modells möglich (siehe Kapitel 5.4). Da-

durch ließen sich einige Modelle auch ohne Anwendung auf dem Testdatensatz als geeignet oder weniger geeignet identifizieren. Die finale Evaluation des Klassifikators basierte letztlich lediglich auf den Ergebnissen des Gesamtmodells bestehend aus Segmentierer und Klassifikator. Dazu wurde das Challengr-Inferenznetzwerk der KiTS21-Challenge angepasst. Äquivalent zur Datenvorbereitung des Klassifikators wurden die Ausgabelabels des Segmentierers auf die erkannten Läsionen reduziert, sodass diese einzeln in den Klassifikator gegeben werden konnten. Da es sich hier um die Inferenz handelt, wurde anders als in der Datenvorbereitung an dieser Stelle der Testdatensatz verwendet. Wie erwähnt, wurden die Labels um das Originalbild und das euklidische Distanzbild erweitert. Für dieses Eingabebild bestimmt der Klassifikator die Klasse (Hintergrund, Niere, Tumor, Zyste) mit der höchsten Wahrscheinlichkeit und basierend auf dem Ergebnis des Klassifikators werden die Ausgabelabels des Segmentierers angepasst. Durch diese Anpassungen lassen sich die Kaskaden-Modelle mittels Challengr vergleichen und deren Mehrwert konnte evaluiert werden (siehe Kapitel 5.4).

4.5.4 Modell

AUTOR*IN: LENA PHILIPP

Im Folgenden werden Details zu dem Modell zusammengefasst, dass bei der KiTS21 Challenge eingereicht wurde. Basierend auf den Erkenntnissen aus den vorangegangenen Experimenten mit den Daten der KiTS19 Challenge, fiel die Wahl der Architektur auf das 3D-U-ResNet. Die Erklärung dafür sowie die Erklärungen für die meisten Entscheidungen finden sich in Kapitel 4.2.3. Ingesamt gab es in diesem Modell circa 1.3 Millionen lernbare Parameter.

- Anzahl an Leveln: 3
- Anzahl an Convolutionschichten pro Residualblock: 2
- Kernelgröße der Convolutionschichten: ((3,3))
- Paddingform: Valid Padding

- Anzahl an Filtern in den Convolutionschichten des ersten Levels: 32
- Normalisierungsform: Batchnormalisierung
- Dropout: 0.2
- Aktivierungsfunktion: ReLU
- Softmax: aktiviert
- Anzahl an Kanälen des Eingabebilds: 1
- Anzahl an Ausgabekanäle: 4

Für die Vorverarbeitung wurden die unter Kapitel 4.5.2 beschriebenen Strategien wie Data Augmentation und Oversampling genutzt. Die konkrete Konfiguration der Parameter für dieses Modell wurde wie folgt gewählt:

- Padding: $21 \times 21 \times 21$
- Patchsize: $32 \times 32 \times 32$
- Voxelsize: $1,5 \times 1,5 \times 1,5$
- Batchsize: 15

Bei dem Schritt der Nachverarbeitung wurde der Ansatz aus Connected Components Niere (Kapitel 4.5.3.2) und MajorityVote (Kapitel 4.5.3.3) der Läsionen gewählt. Das Training allgemein betreffend wurde ein Split in 70% Training / 10% Validierung / 20% Test gewählt, außerdem wurde early stopping verwendet und nicht das letzte, sondern das beste Modell ausgewählt. Die Validierung während des Trainings erfolgt nach 800 Trainingsiterationen auf der Grundlage von 500 Patches aus dem Validierungsdatensatz, aufgeteilt in Validierungsbatches aus 50 Patches. Die Dauer des Trainings betrug 4 Tage und das beste Modell wurde nach 169601 Iterationen bestimmt. Der Verlauf des Trainings ist abgebildet in Abbildung 4.15.

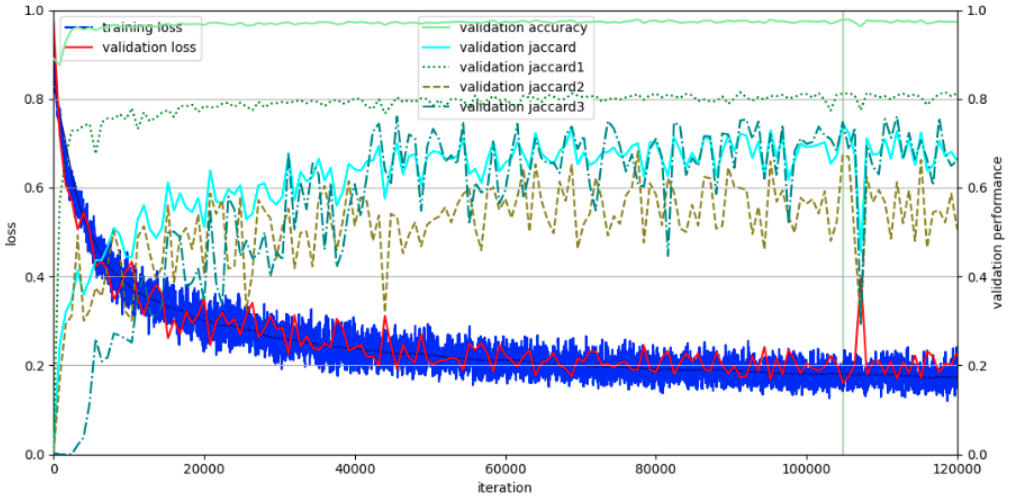


Abbildung 4.15: Die Jaccard-Werte 1 bis 3 entsprechen Niere, Tumor und Zyste

5 Auswertung

5.1 Deep Medic

AUTOR*IN: MARCEL PLUTAT

Zum Testen der Deep Medic Implementierung, aus Kapitel 4.2.4, wurden einige Trainings mit den Standardwerten gestartet. Die Trainings zur Validierung der Implementierung und Überprüfung der verschiedenen Parameter von Deep Medic bilden die Basis für die Deep Medic Experimentenserie. Die Experimentenserie für Deep Medic ist auf den Daten der KiTS19 Challenge erfolgt. Neben dem Test der Implementierung war ein weiteres Ziel die Bestimmung der besten Parameter für Deep Medic um einen möglichst hohen Kidney- und Tumor Dice auf den KiTS Daten zu erzielen. Ein gutes Modell für die KiTS19 Challenge zu finden, war eines der ursprünglichen Projektziele und wurde daher auch für die Deep Medic Architektur verfolgt.
Das erste Experiment befasst sich mit der Eingabegröße der Patches. Die Referenz hat eine Eingabe von 25^3 und 19^3 für die normale und niedrige Auflösung. Für die Experimentenserie wurden die Eingaben 34^3 und 22^3 sowie 43^3 und 25^3 als Auflösungen betrachtet. Die Auflösungen wurden gewählt, da sich die Ausgabegröße bei diesen Werten jeweils verdoppelt. Tabelle 5.1 zeigt die Referenz in Zeile 1 und die größeren Eingaben in den Zeilen 2 und 3. Dabei ist deutlich erkennbar, dass eine höhere Auflösung in einem höheren Dice Wert, sowohl für Niere als auch Tumor, resultiert. Für die restliche Auswertung wurden daher nur größere Eingabe Patches verwendet.

#	Normale Auflösung	Niedrige Auflösung	Nieren Dice	Tumor Dice
1	25,25,25	19,19,19	0.861	0.291
2	34,34,34	22,22,22	0.884	0.327
3	43,43,43	25,25,25	0.904	0.338

Tabelle 5.1: Deep Medic Vergleich der Eingabegrößen

Im nächsten Schritt wurde die Voxelsize betrachtet. Bisher wurde eine Voxelsize von $1,5mm$ für alle Trainings verwendet. Dazu wurde ein neues Training mit der Voxelsize $1,0mm$ gestartet. Die niedrigere Voxelsize hat allerdings keine Verbesserung der Dice Werte gebracht und wurde in der Folge nicht weiter überprüft. An dieser Stelle sei angemerkt, dass höhere Voxelsizes nicht überprüft wurden, aber in den Trainings der anderen Architekturen, wie dem U-ResNet, relativ gute Ergebnisse geliefert haben. Eine zukünftige Überprüfung könnte an dieser Stelle erneut ansetzen und nach besseren Parametern für die Voxelsize suchen.
Des Weiteren wurde der Einfluss der Anzahl der Convolutional Layer auf die Ergebnisse überprüft. Standardmäßig hat Deep Medic acht Convolutional Layer. Tabelle 5.2 zeigt den Vergleich zwischen acht, zehn und zwölf Convolutional Layern. Aufgrund der unterschiedlichen Anzahl von Convolutional Layern haben alle Modelle verschiedene Eingabegrößen. Diese wurden allerdings so gewählt, dass alle Modelle dieselbe Ausgabegröße haben. In Tabelle 5.2 ist zu erkennen, dass die Erhöhung auf zehn Convolutional Layer eine Verbesserung ergibt. Eine weitere Steigerung auf zwölf Layer hat keinen positiven Effekt gehabt. Folglich werden zehn Convolutional Layer in den weiteren Experimenten verwendet. In einem weiteren Test für zehn Convolutional Layer wurde die Eingabe soweit verringert dass die Ausgabegröße 21^3 ergibt. Dieses Modell zeigt Modell 5.1 in Tabelle 5.2. Modell 5.1 hat Modell 5 im Tumor Dice deutlich übertroffen und wird in der Folge als bestes Modell betrachtet.

#	Filter pro Layer	Nieren Dice	Tumor Dice
4	30, 30, 40, 40, 40, 40, 50, 50	0.904	0.338
5	30, 30, 30, 40, 40, 40, 40, 50, 50, 50	0.923	0.342
5.1	30, 30, 30, 40, 40, 40, 40, 50, 50, 50	0.892	0.410
6	30, 30, 30, 40, 40, 40, 40, 40, 40, 50, 50, 50	0.908	0.291

Tabelle 5.2: Deep Medic Vergleich der Anzahl an Convolutional Layern

Die Kernel Size wurde in einem Training auf 5^3 erhöht. Die Konsequenz daraus war allerdings, dass die Eingaben sehr groß gewählt werden mussten und die Batch Size verringert werden musste. Aus diesem Grund wurde für dieses Training ebenfalls Gradient Accumulation über 3 Iterationen verwendet. Die Ergebnisse sind vergleichbar mit denen von Modell 5 aus Tabelle 5.2, übertreffen Modell 5.1, das aktuell beste Modell, allerdings nicht.

In einem weiteren Schritt wurde das bisher beste Modell, Modell 5.1, um Data Augmentation im Preprocessing erweitert. Als Data Augmentation wurde Rotation um $-30°$, $0°$ und $30°$ verwendet. Data Augmentation hat keinen Mehrwert gegenüber dem Training ohne Data Augmentation geliefert und wurde daher nicht weiter betrachtet. Das Training mit Data Augmentation war außerdem pro Iteration deutlich langsamer als ohne Data Augmentation. Die Laufzeit war daher ein weiteres Argument gegen die Verwendung von Data Augmentation.

Zuletzt wurde Postprocessing auf Modell 5.1, dem bisher besten Modell, angewandt. Als Postprocessing wurde das *Connected Components* Modul verwendet, welches für die KiTS21 Challenge verwendet wurde um die Netzausgabe zu verbessern. Auch an dieser Stelle verbessert dieses Postprocessing die Dice Werte auf $0,918$ für den Nieren Dice und den Tumor Dice auf $0,560$.

Der zweite Teil der Experimentenserie hat sich mit den Residual Verbindungen für Deep Medic befasst. Zuerst wurden Deep Medic und Deep Medic mit Residual Verbindungen miteinander verglichen. Beide Trainings waren identisch bis auf die Residual Verbindungen im zweiten Training. Tabelle 5.3 zeigt Deep Medic als Modell 7 und Deep Medic mit Residual Verbindungen als Modell 8. Dabei wird deutlich, dass die Standard-

parameter für Deep Medic mit Residual Verbindungen die Dice Werte verschlechtern. In der Folge werden die Parameter der Residual Verbindungen angepasst um das beste Modell zu finden.

#	# Residual Layer	Indices	Offset	Nieren Dice	Tumor Dice
7	-	-	-	0.861	0.291
8	3	4,6,8	2	0.534	0.221

Tabelle 5.3: Deep Medic Vergleich mit Residual Verbindungen

Als Basis für die Auswertung der Residual Verbindungen wird das beste Modell aus Teil 1 der Experimentenserie, Modell 5.1, verwendet. Die Anzahl der Residual Verbindungen und der damit verbundene Offset werden nun überprüft. Der Vergleich der Modelle mit mehr Residual Verbindungen ist in Tabelle 5.4 aufgelistet. Modell 9 zeigt eine Architektur mit 4 Residual Verbindungen und einem Offset von 2, während Modell 10 eine Architektur mit 7 Residual Verbindungen und einem Offset von 1 ist. Das Modell in Zeile 1 liefert die besseren Ergebnisse, ist allerdings immer noch schlechter als ein Modell ohne Residual Verbindungen.

#	Indices	Offset	Nieren Dice	Tumor Dice
9	4,6,8,10	2	0.807	0.324
10	4,5,6,7,8,9,10	1	0.843	0.304

Tabelle 5.4: Deep Medic Vergleich der Anzahl an Residual Verbindungen

Abschließend lässt sich festhalten, dass größere Eingaben und mehr Convolutional Layer zu den besten Ergebnissen führen. Dagegen bringen Data Augmentation und Residual Connections keinen großen Mehrwert, da keine Verbesserungen vorliegen. Zuletzt hat das Postprocessing durch *Connected Components* die Segmentierung effektiv verbessert und zu den besten Ergebnissen geführt.

5.2 AU-Net

AUTOR*IN: MARKUS RINK

Die AU-Net Experimentenserie soll untersuchen, ob die Architektur bessere Ergebnisse liefert als vergleichbare U-Net Netzwerke und ob die Salienz Karten gelernt werden. Für letzteres wurde mit und ohne Deep Supervision trainiert, wobei sich die Implementierung gegenüber der aus dem Paper unterscheidet (siehe Kapitel 3.4.2).

Trainiert wurden 5 Netze mit folgenden Eigenschaften:

- Preprocessing
  - Voxelsize Resampling auf $2,5mm \times 2,5mm \times 1mm$
  - Untere Schranke der HU Werte auf -1024HU
  - Oversampling mit 50% enthält Tumor und 50% nur Niere
- Architektur
  - 3 Level
  - zwei Attention Gates pro Skip Connection
  - 16 Filter pro Convolution, bzw. 8 Filter bei aktiver Deep Supervision, wobei die Filteranzahl bei jedem Level verdoppelt wird
  - zwei $3 \times 3 \times 3$ Convolutions pro Level
  - Batch Normalization
  - Lernrate von $0,0001$ (ausgenommen Netz 5 mit Novograd [58, 59] und $0,005$)
  - Batchsize 14, bzw. 10 bei aktiver Deep Supervision
  - Eingabepatches $64 \times 64 \times 64$ (inkl. $40 \times 40 \times 40$ Padding)

Tabelle 5.5 fasst die Trainingsergebnisse zusammen, wobei Netz 4 aufgrund technischer Probleme nicht ausgewertet werden konnte. Die AU-Nets konnten eine leichte Verbesserung in beiden Dice Scores gegenüber dem U-Net erzielen, mit Außnahme von Netz 3, bei dem die Tumor Segmentierung nicht konvergiert ist. Aufgrund der Verwendung von Deep Supervision wurden die Filteranzahl und Batchsize verkleinert und außerdem trainierte das Netz 3 vergleichsweise kurz mit 36800 Iterationen. Daher könnte hier ein zu früher Trainingsabbruch die Ursache für den

unterdurchschnittlichen Tumor Dice sein. Am Deutlichsten lässt sich der Einfluss vom Oversampling sehen, auf welche die Verbesserung in der Tumorsegmentierung zurückzuführen ist (vgl. 5.3.5).

Die AU-Nets haben durchweg schlechtere Dice Scores als die betrachteten U-ResNets. Bei den Unterschieden fällt auf, dass die U-ResNets mehr Filter verwenden und isotrope Eingaben zum Training nutzen.

	Deep Supervision	Dropout	Iterationen	Filter	Batch	Dice Niere	Dice Tumor
			AU-Nets				
1		0.25	40800	16	14	0.852	0.236
2		0.25	147200	16	14	0.877	0.297
3	x	0.25	36800	8	10	0.815	0.004
4	x	0	2336522	16	8	?	?
			U-Nets				
5		0.2	198401	16	2	0.827	0.000
6		0.2	1472001	16	2	0.849	0.088
7		0.2	1472001	16	2	0.889	0.377
			U-ResNets				
8		0	499201	32	15	0.929	0.439
9		0	108801	32	15	0.933	0.436
10		0	144801	32	8	0.936	0.438
11		0	140801	64	8	0.931	0.463

Tabelle 5.5: KiTS19 AU-Netze

Anmerkung zu U-Nets: Netz 5 hat eine Voxelsize von $2mm \times 2mm \times 2mm$*, Netz 6* $2.5mm \times 2.5mm \times 2.5mm$ *und Netz 7* $4mm \times 4mm \times 2.5mm$*. Im Preprocessing von Netz 7 wird das selbe Oversampling verwendet, wie in dieser Experimentenserie.*
Anmerkungen zu U-ResNets: Alle haben eine Voxelsize von $2.5mm \times 2.5mm \times 2.5mm$ *und kein Oversampling.*

Zum Auswerten der Salienz Karten wurden die Aktivierungen der α Layer (vgl. Kapitel 3.4.2) für ein Eingabebild aus der Testmenge betrachtet. Bis auf punktuelle Stellen wurde gelernt die Skip Connection zu ignorieren. Dies könnte in einem einzelnen Convolutional Layer, wie beim U-Net, genauso gelernt werden, wodurch das AU-Net in diesem Fall einen rechnerischen Overhead beim Lernen hat.

Die im AU-Nets Paper [40] gezeigten Salienz Karten konnten also nicht reproduziert werden. Der Hauptgrund dafür ist vermutlich die Implementierung von Deep Supervision. Diese basiert weder auf dem U-Net, noch

dem AU-Net und sollte daher in RedLeaf geändert werden oder einen erklärenden Kommentar im Code bekommen.

5.3 U-ResNet

AUTOR*IN: NIKLAS AGETHEN

Die Auswertung der Trainingsserien mit U-ResNet Modellen beinhaltet sowohl Trainings mit den Daten der KiTS19 als auch der KiTS21 Challenge. Die im Kapitel 4.2.3 angesprochenen Trainingsserien lieferten Erkenntnisse über die Auswirkungen verschiedener Architekturparameter und Preprocessing-Konfigurationen, die im weiteren Verlauf des Kapitels näher beschrieben werden. Neben der Analyse der gewählten Einstellungen galt es zudem, diese miteinander zu vergleichen und ein möglichst bestes Modell zu identifizieren. Für den Vergleich verschiedener Modelle wurde die Challengr Plattform und die für die jeweilige Challenge relevanten Metriken verwendet. Die Metriken der KiTS19 Challenge bilden der Nieren-Dice sowie der Tumor-Dice. Bei der KiTS21 Challenge werden die **Hierarchical Evaluation Classes (HECs)** für die Evaluation angewandt [1]. Dieser hierarchische Ansatz fasst zunächst mehrere Klassen in einer Metrik zusammen und bildet dann einzelne Klassen in einer separaten Metrik ab. Folgende Metriken stellen die HECs der KiTS21 Challenge dar:

1. Kidney-and-Masses-Dice *(Niere + Tumor + Zyste)*
2. Kidney-and-Masses-Surface-Dice
3. Kidney-Mass-Dice *(Tumor + Zyste)*
4. Kidney-Mass-Surface-Dice
5. Tumor-Dice
6. Tumor-Surface-Dice

Bei der Auswertung und dem Vergleichen der Modelle muss die fehlende statistische Signifikanz berücksichtigt werden. Unterschiede der Modelle

sind möglicherweise zufallsbedingt durch die Initialisierung der Gewichte des Neuronalen Netzes.

5.3.1 Netztiefe und Voxelsize

AUTOR*IN: NIKLAS AGETHEN

Die Netztiefe des U-ResNet Modells kann über die Anzahl an Leveln sowie die Anzahl an Convolutionschichten pro Level reguliert werden. Zu Beginn der Trainingsserien wurden daher einige Trainings gestartet, die sich lediglichen in diesen beiden Parametern unterschieden. Die übrigen Parameter wurden gemäß der im Kapitel 4.2.3 vorgestellten Basiswerte definiert. Somit beziehen sich die unten dargestellten Trainingsergebnisse auf Modelle mit der Voxelsize von $2,5mm$.

#	Level	Conv. pro Level	Nieren Dice	Tumor Dice
1	2	2	0.881	0.334
2	2	5	0.921	0.436
3	4	2	0.914	0.339
4	4	4	0.911	0.420
5	5	2	0.907	0.331

Tabelle 5.6: U-ResNet Vergleich der Netztiefe

Wie in der Tabelle 5.6 zu sehen, weichen die Nierendice Werte weniger stark voneinander ab wie die Tumordice Werte. Die vergleichsweise kleinen Netze mit zwei Leveln erreichen einen ähnlich hohen Wert wie die Netze mit mehr Leveln, wenn die Anzahl an Convolutionschichten pro Level erhöht wird. In Bezug auf das rezeptive Feld lässt sich aus der Experimentenserie ablesen, dass ein Netz mit einem rezeptiven Feld von 44 Voxeln (2 Level, 5 Convolutionschichten pro Level) für die Voxelsize von $2,5mm$ ausreichend ist. Kleinere rezeptive Felder, wie etwa das von 20 Voxeln (2 Level, 2 Convolutionschichten pro Level) erzielen selbst bei der vergleichbar hohen Voxelsize von $2,5mm$ deutlich schlechtere Nierendice-Ergebnisse und sind daher als weniger geeignet anzusehen. An dieser Stelle wurde zudem die Größe der zu segmentierenden

Struktur aus Kapitel 4.5.1 von durchschnittlich etwa $80mm \times 80mm$ (x, y; *transversal*) zur Analyse hinzugezogen. Diese Durchschnittsgröße entspricht bei einer Voxelsize von $2,5mm$ etwa $32mm \times 32mm$. Daher scheint sich die Annahme hier zu bestätigen, dass das rezeptive Feld mindestens der Größe der zu segmentierenden Struktur entsprechen sollte. Das rezeptive Feld von 20 scheint hier für die Segmentierung der Niere nicht auszureichen.

Aus den Ergebnissen lässt sich zudem ablesen, dass die Netze mit mehr Convolutionschichten pro Level einen signifikant besseren Tumordice erzielen als die Netze mit lediglich zwei Convolutionschichten. Diese Beobachtung konnte jedoch in weiteren Trainingsserien durch den Vergleich mehrerer Netze mit unterschiedlich vielen Convolutionschichten pro Level widerlegt werden. Aufgrund der geringeren Anzahl an zu paddenden Voxeln, wurden daher für die weiteren Trainings hauptsächlich Architekturen mit dem Standardwert von zwei Convolutionschichten pro Level verwendet.

Nach dem Wechsel zu den KiTS21 Daten starteten wir eine weitere Trainingsserie bezüglich der Netztiefe von 3D U-ResNets in Kombination mit unterschiedlichen Voxelsizes. Dabei galt es erneut das beste Modell unterschiedlicher Levelmengen für die einzelnen Voxelsizes zu finden. Gerade das Verringern der Voxelsize versprach aufgrund der höheren Auflösung eine Verbessung der Ergebnisse. Wie im Kapitel 4.5.1 beschrieben, wurde ein Großteil der Originalbilder in feinerer Voxelsize als $2,5mm$ aufgenommen. Da sich durch das Anpassen der Voxelsize die Größe der zu segmentierenden Struktur verändert, ist ggf. das gleichzeitige Anpassen der Netztiefe und somit der Größe des rezeptiven Felds notwendig. Daher wurde diese beiden Parameter in einer Trainingsserie gebündelt.

#	Level	Voxelsize	Mean KiTS21 Metrics
1	2	2.5	0.614
2	3	2.5	0.634
3	3	1.5	0.681

Tabelle 5.7: U-ResNet Vergleich der Netztiefe und Voxelsizes

Zunächst untersuchten wir unterschiedliche Voxelsizes mit zwei- und drei-

Level U-ResNets. Obwohl das rezeptive Feld des zwei Level U-ResNets mit 20 Voxeln zuvor als zu klein eingeschätzt wurde, weicht der Durchschnitt aller KiTS21 Metriken nur minimal von den drei Level Pendants ab (siehe Tabelle 5.7). Bei den drei Level Netzen zeigt sich zudem, dass sich die feinere Voxelsize positiv auf die Metriken auswirkt.

#	Level	Voxelsize	Mean KiTS21 Metrics
1	3	1.5	0.760
2	3	1	0.634
3	4	1.5	0.744
4	4	1	0.709

Tabelle 5.8: U-ResNet Vergleich der Netztiefe und Voxelsizes

Nach dem Erweitern des Postprocessings (siehe Kapitel 4.5.3) wurden weitere Trainings unterschiedlicher Tiefe und Voxelsize gestartet. Dabei wurde untersucht, ob sich durch das Hinzufügen weiterer Level und feinerer Voxelsizes, die Ergebnisse weiter verbessern konnten. Wie in der Tabelle 5.8 zu sehen, haben sich die durchschnittlichen Metriken allerdings nicht weiter verbessert. Da die drei Level Netze leicht bessere Ergebnisse lieferten als die vier Level Netze, wurden bei den weiteren Experimenten der KiTS21 Challenge in erster Linie U-ResNets mit drei Leveln verwendet. Zumal die drei Level Netze aufgrund des geringeren Paddings und des GPU-Speicherlimits mehr Freiheiten in der Auswahl der Filter-, Patch- und Batchgrößen erlauben. Auf Architekturen mit fünf Leveln wurde ebenfalls aufgrund des GPU-Speicherlimits verzichtet, da das Padding von 186 Voxeln pro Dimension (bei zwei $3 \times 3 \times 3$ Convolutions pro Level) eine enorme Beschränkung der übrigen speicherrelevanten Parameter nachsichziehen würde. Außerdem fokussierten sich die weiteren Trainings hauptsächlich auf die Voxelsize von 1,5mm.

5.3.2 KiTS19 2D vs. 3D

AUTOR*IN: TIMO GÜNNEMANN

Als wir mit den Trainingsserien für das U-ResNet auf dem KiTS19 Datensatz begonnen haben, haben wir die Einstellungen vom U-Net übernommen. Somit haben wir viele Netze mit einer Patchsize von 100×100, einer Batchsize (BS) von 15 und 64 Filtern trainiert. Da es bei diesen zweidimensionalen Netzen keine Probleme mit dem Speicher des GPU-Clusters gab, waren wir nicht gezwungen an den Parametern etwas anzupassen. In der Tabelle 5.9 ist ein Teil unserer Ergebnisse dieser Netze zu sehen. Die unterschiedlichen Ergebnisse hängen damit zusammen, dass die Modelle mit verschiedenen Parametern trainiert wurden, auf die wir in diesem Teil nicht weiter eingehen werden.

#	Level	Patchsize	BS	Filtersize	Nieren Dice	Tumor Dice
1	2	100x100	15	64	0.922	0.478
2	4	100x100	15	64	0.914	0.359
3	4	104x104	15	64	0.912	0.402
4	5	100x100	15	64	0.907	0.331

Tabelle 5.9: U-ResNet 2D KiTS19

Nachdem wir die Dimensionalität auf drei erhöhten, konnten wir unsere vorherigen Einstellungen nicht beibehalten, da die Netze mehr GPU-Speicher benötigten als auf dem Cluster zur Verfügung stand. Daraufhin haben wir in verschiedenen Konfigurationen die Patchsize, die Batchsize und die Filtersize angepasst. Die Tabelle 5.10 zeigt beispielhaft ein Ausschnitt von den Konfigurationen. Es war notwendig ein Gleichgewicht mit den Parametern zu schaffen. Das bedeutet, wenn wir eine höhere Patchsize haben wollten, mussten wir dafür die Batchsize und / oder die Filtersize reduzieren. Wollten wir stattdessen eine höhere Batchsize und / oder Filtersize war es notwendig die Patchsize zu reduzieren. Außerdem haben wir non-cubic Patchsizes verwendet um dem Speicherproblem entgegenzuwirken. Dafür haben wir die Größe des Patches z.B. in der z-Dimension reduziert, wodurch die einzelnen Patches kleiner sind

und somit mehr Speicher zu Verfügung steht für z.B. eine höhere Batchsize.

#	Level	Patchsize	BS	Filtersize	Nieren Dice	Tumor Dice
1	2	92x92x92	2	64	0.919	0.467
2	3	96x96x8	8	32	0.936	0.438
3	3	60x60x8	8	64	0.931	0.463
4	3	20x20x20	15	32	0.929	0.439
5	4	36x36x36	2	32	0.907	0.436

Tabelle 5.10: U-ResNet 3D KiTS19

Die zweidimensionalen Netze erzielten grundsätzlich vergleichbar gute Ergebnisse, sowohl was den Nieren als auch den Tumor Dice anbelangt. Das beste Netz hat sogar einen besseren Tumor Dice als das beste dreidimensionale Modell. Aber das war auch eine Ausnahme, im Schnitt waren die dreidimensionalen Modelle besser. Natürlich muss man hier auch wieder anmerken, dass die Ergebnisse nicht statistisch signifikant sind. Außerdem konnten wir aufgrund des Speichers nicht die gleichen Parameter wie für die zweidimensionalen Netze nehmen.

5.3.3 KiTS21 3D

AUTOR*IN: TIMO GÜNNEMANN

Bei der KiTS21 Challenge haben wir keine Netze mit zwei Dimensionen trainiert, da wir bei den Experimentenserien auf dem KiTS19 Datensatz gesehen haben, dass die dreidimensionalen Netze im Schnitt bessere Ergebnisse liefern. Es bestand weiterhin das Problem des begrenzten GPU-Speichers, das insbesondere bei dreidimensionalen Netzen eine Hürde darstellte. Wir haben somit am Ende mit relativ kleinen Patchgrößen trainiert, um trotzdem nicht zu kleine Batch- und Filtersizes zu haben.

#	Level	Patchsize	BS	Filtersize	Mean KiTS21 Metrics
1	2	38x38x38	5	64	0.700
2	2	46x46x46	10	64	0.670
3	2	92x92x92	2	63	0.646
4	3	32x32x32	15	32	0.785
5	3	36x36x36	20	24	0.757
6	4	36x36x36	3	32	0.781
7	4	36x36x36	6	16	0.760

Tabelle 5.11: U-ResNet 3D KiTS21

5.3.4 Data Augmentation

AUTOR*IN: LENA PHILIPP

Hinsichtlich des Einflusses von Data Augmentation auf die Leistung wurden aus 115 U-ResNet Modellen 60 ausgewählt. Da die gleichen Modelle durch verschiedenes Postprocessing mehrfach im Datensatz vorhanden waren, wurde nach dem Ablageverzeichnis und dem KiTS21 Mean Metrics Score sortiert. Anschließend wurden Duplikate entfernt und nur das jeweils beste Modell pro Ablageverzeichnis behalten. Von diesen 60 wurden 37 mit Data Augmentation trainiert. Für die Auswertung werden die Dice Werte der KiTS21 Challenge sowie die der einzelnen Klassen verwendet. Für das Vergleichen von Werten werden Boxplots genutzt und der Median betrachtet, da dieser weniger sensibel auf Ausreißer reagiert, die besonders bei kleinen Vergleichsgruppen vorliegen.

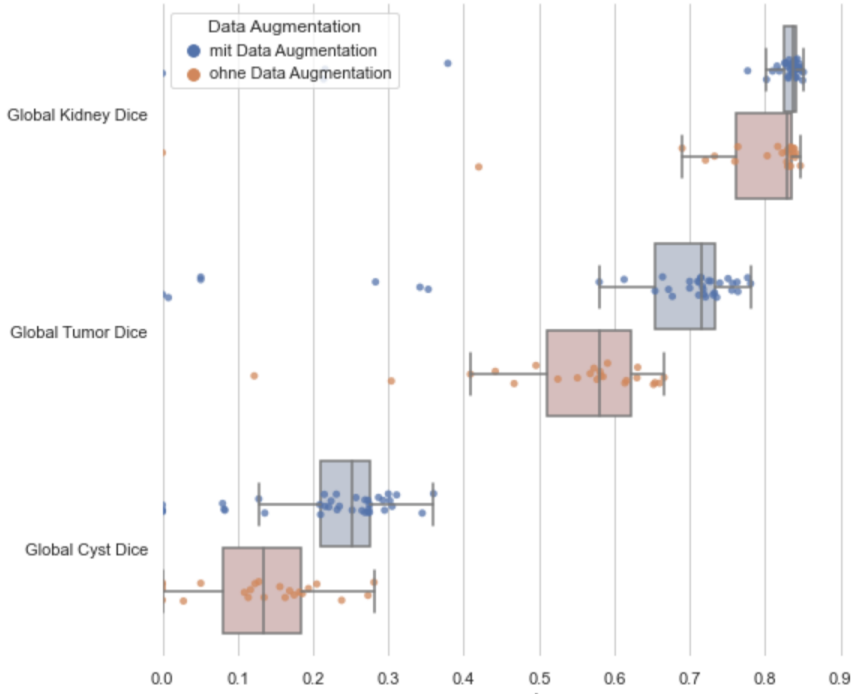


Abbildung 5.1: Einfluss von Data Augmentation auf den Dice von Niere, Tumor und Zyste

Schließlich wurde der Einfluss auf die einzelnen Dice Werte der Klassen untersucht, wie mit den Boxplots der Abbildung 5.1 visualisiert wird. Diese zeigen die Dice Werte der jeweiligen Klassen mit und ohne Data Augmentation.

Metrik	Mit Data Augmentation	Ohne Data Augmentation
Niere Dice Median	0.837	0.829
Tumor Dice Median	0.715	0.581
Zyste Dice Median	0.252	0.135

Tabelle 5.12: Vergleich der Median Dice Werte der einzelnen Klassen in Bezug auf Data Augmentation

Gemeinsam mit der Tabelle 5.12 zeigt sich die Tendenz, dass Data Augmentation die Segmentierung von Tumor und Zyste verbessert. Bei der Niere scheint der Einfluss geringer.

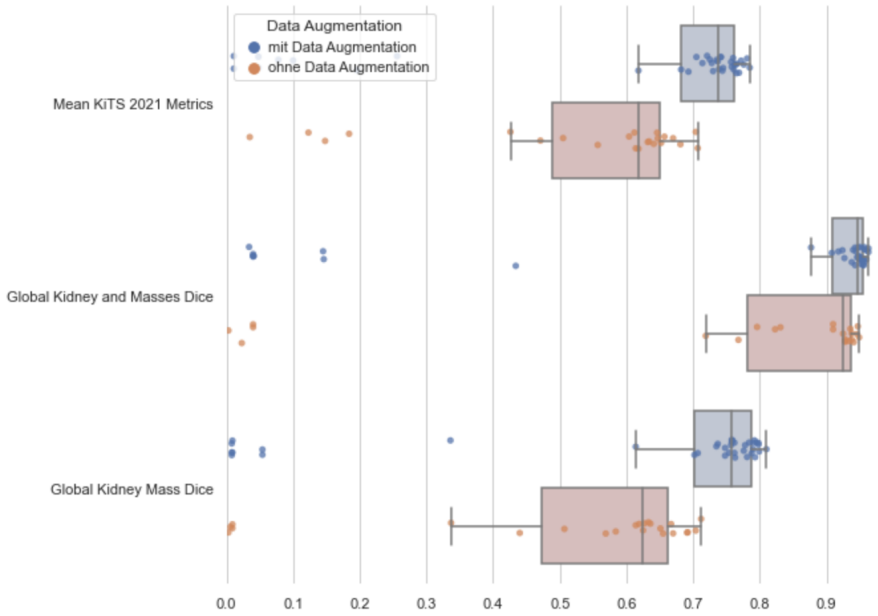


Abbildung 5.2: Einfluss von Data Augmentation auf die zusammengesetzten Werte KiTS21 Mean Metrics, Kidney Mass Dice und Kidney and Masses Dice

Durch die Abbildung 5.2 wird sichtbar, wie sich die beschriebenen Effekte auf die zusammengesetzten Metriken auswirken. Generell wird eine Verbesserung durch Data Augmentation deutlich, außerdem nimmt die Streuung der Ergebnisse ab. Die größte Veränderung lässt sich für den Kidney Mass Dice beobachten, dieser wird am stärksten durch Veränderung des Zysten und Tumor Dice beeinflusst.

5.3.5 Oversampling

AUTOR*IN: LENA PHILIPP

Um den Einfluss von Oversampling zu untersuchen wurden 60 verschiedene U-ResNet Modelle mit Postprocessing mit und ohne SPS untersucht. Diese Modelle wurden auf die gleiche Weise wie in Kapitel 5.3.4 ausgewählt. Das Verhältnis der Klassen für das Oversampling, wie in Kapitel 4.5.2 beschrieben, ist für die Modelle mit SPS gleich. Allerdings wurden insgesamt weniger Modelle ausgewertet, bei denen für das Training kein SPS genutzt wurde. Unter den 60 ausgewerteten Modellen wurden

nur bei 10 kein Oversampling genutzt, da sich bereits früh beim Training zeigte, dass Netze ohne SPS nicht konvergierten und der Jaccard Wert bei der Validierung für die Zyste bei 0 blieb. Die meisten dieser Netze wurden auf Grund der schlechten Ergebnisse auf den Validierungsdaten nicht für die Testdaten ausgewertet. Dadurch ist es schwer den Einfluss dieses Parameters zu untersuchen, jedoch werden Tendenzen deutlich.

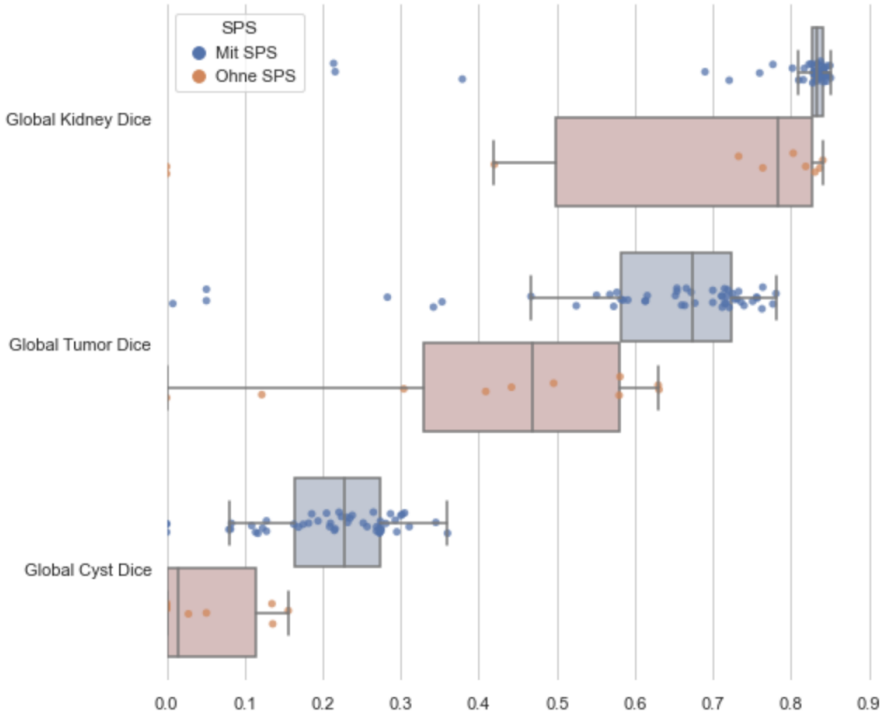


Abbildung 5.3: Einfluss von Oversampling auf den Dice von Niere, Tumor und Zyste

Es zeigt sich in Abbildung 5.3, dass für die Zyste der Einfluss des Oversamplings größer zu sein scheint als bei den anderen beiden Klassen. Wird beim Training kein Oversampling genutzt, bleibt der Zysten Dice vergleichsweise niedrig. Ähnliches lässt sich für die Klasse Tumor beobachten. Auch hier sind die Dice Werte wenn kein SPS beim Training genutzt wurde unter den Ergebnissen der Modelle mit Oversampling. Dass das Ausgleichen des Ungleichgewichts der Klassen trotzdem besonders für das Erkennen der Zyste relevant ist, zeigt sich in der bereits erwähnten Beobachtung, dass die Trainings ohne SPS für die Zyste vermehrt nicht konvergiert sind. Für die Niere lässt sich der Effekt des Oversampling am

geringsten beobachten, da Modelle mit oder ohne Nutzung des SPS gute Kidney Dice Werte erlangen. Die Stärke des Effektes spiegelt somit die Seltenheit der Klasse wieder.

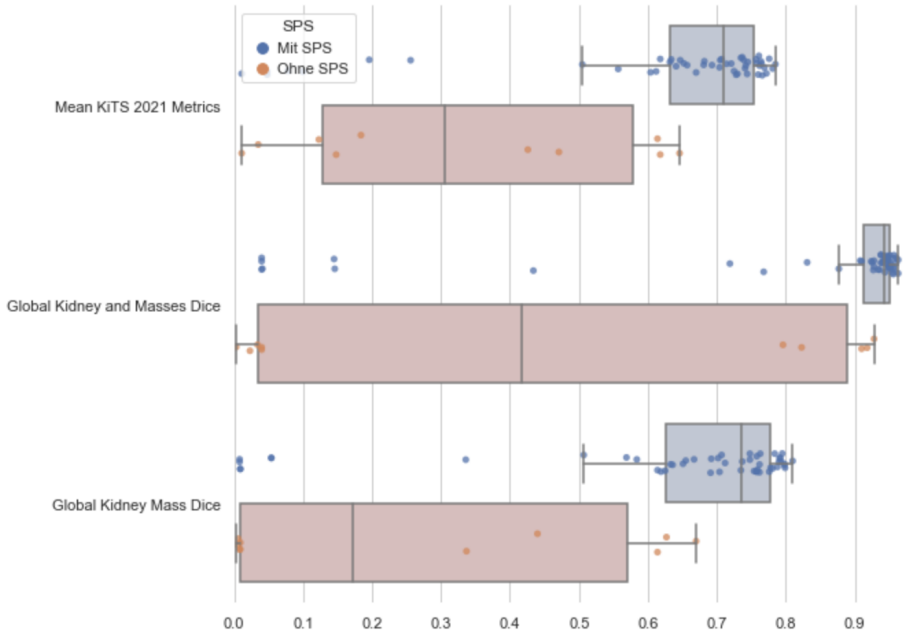


Abbildung 5.4: Einfluss von Oversampling auf die zusammengesetzten Werte KiTS21 Mean Metrics, Kidney Mass Dice und Kidney and Masses Dice

Für die zusammengesetzten Metriken der Challenge deutet sich darüber hinaus an, dass durch das Verwenden von Oversampling die Streuung abnimmt. Dies kann auch durch das Ungleichgewicht der ausgewerteten Modelle erklärt werden. Abbildung 5.4 zeigt, dass der Median für alle drei Metriken weit auseinander liegt und diese Modelle mit Oversampling zu besseren Ergebnissen gekommen sind. Für den Kidney Mass Dice unterscheiden sich die Ergebnisse beider Gruppen am stärksten, was nochmal den Eindruck verstärkt, der sich bereits in Abbildung 5.3 andeutete.

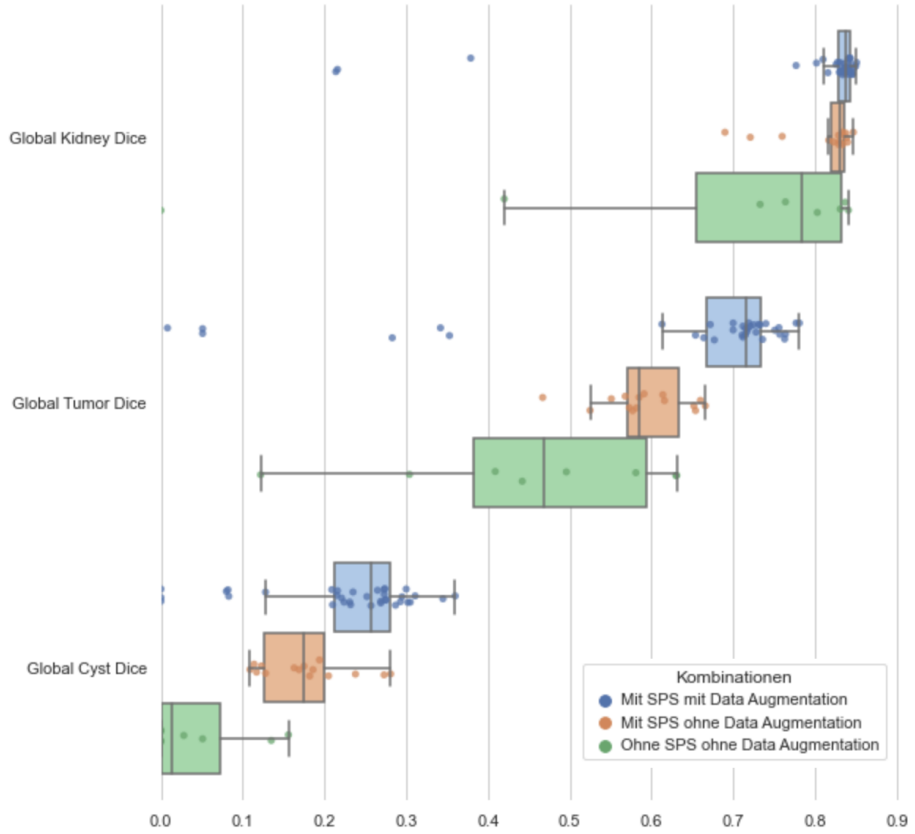


Abbildung 5.5: Einfluss von Oversampling und Data Augmentation in Kombination auf den Dice von Niere, Tumor und Zyste

Abbildung 5.5 zeigt wie sich die Kombination aus Data Augmentation und Oversampling auf die Dice Werte der einzelnen Klassen auswirkt. Als Grundlage dienen die 60 Modelle, die Anteile der jeweiligen Gruppen zeigt die Tabelle 5.13. Da es nur zwei Modelle mit Data Augmentation und ohne SPS gibt, wurde diese Kombination ausgeschlossen. Das Vorhandensein von Data Augmentation und Oversampling verbessert den Median des Tumor Dice von 0.469 auf 0.715 im Vergleich dazu, wenn beides nicht eingesetzt wird. Wichtig ist dabei zu beachten, dass in der ersten Gruppe mehr als viermal so viele Modelle sind und das Ergebnis eine zufällige Beobachtung sein kann, dennoch wird hier eine Tendenz deutlich. Gleiches gilt auch für den Dice der Zyste.

Metrik	Mit Data Augmentation	Ohne Data Augmentation
Mit SPS	35 (58%)	15 (25%)
Ohne SPS	2 (2%)	8 (13%)

Tabelle 5.13: Vergleich der Median Dice Werte der einzelnen Klassen in Bezug auf Data Augmentation

5.4 Postprocessing

AUTOR*IN: LENA PHILIPP

Im folgenden werden die Experimente bezüglich des Postprocessings aufgeführt.

5.4.1 Connected Components einfach

Zunächst wird nur der Schritt des Postprocessings betrachtet, indem die falsch-positiven Artefakte entfernt werden. Dabei wurden insgesamt fünf Modelle ausgewählt. Trainingsparameter wie Architektur, Tiefe oder Voxelgröße, wurden dabei ignoriert. Jedes Modell wurde einmal ohne Postprocessing und mit Postprocessing mit *Connected Components* einfach, wie in Kapitel 4.5.3.1 beschrieben, ausgewertet. Auf zusammengesetzten Metriken *Kidney and Masses* sowie *Kidney Mass* folgen die einzelnen Dice Werte für die drei Klassen Nieren, Tumor und Zyste. Zur visuellen Vergleichbarkeit werden die Boxplots trotz der großen Unterschiede der Dice Werte auf einer einheitlichen Skala von 0 bis 1 abgebildet.

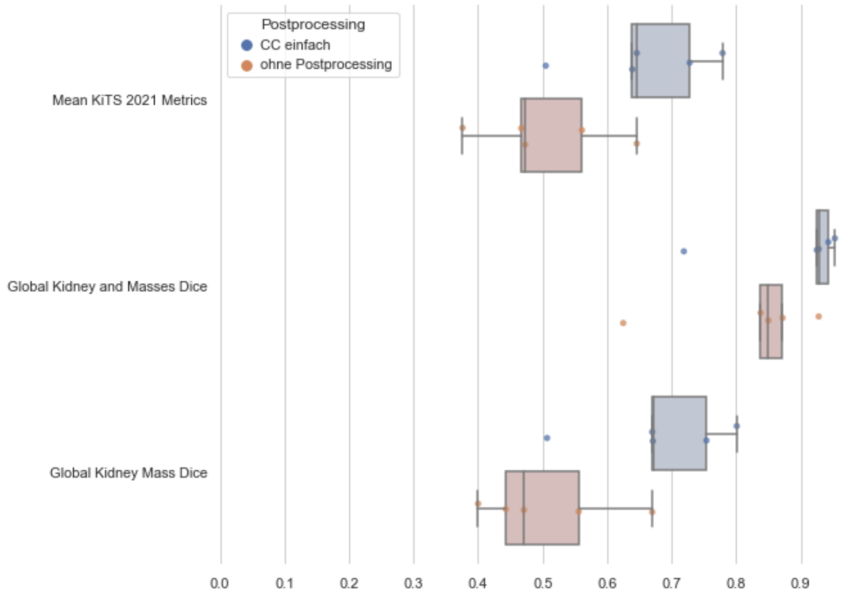


Abbildung 5.6: Vergleich der zusammengesetzten Metriken für Modelle mit Postprocessing CC einfach und ohne Postprocessing

In der Abbildung 5.6 wird deutlich, dass das einfache Postprocessing bereits die Metriken merkbar verbessert. So steigt der Median von dem *Kidney Mass* Dice durch die Nachverarbeitung von 0.471 auf 0.671. Außerdem wird bei dem *Kidney and Masses* Dice die Spannweite des Boxplots sowie der Interquartilsabstand verringert.

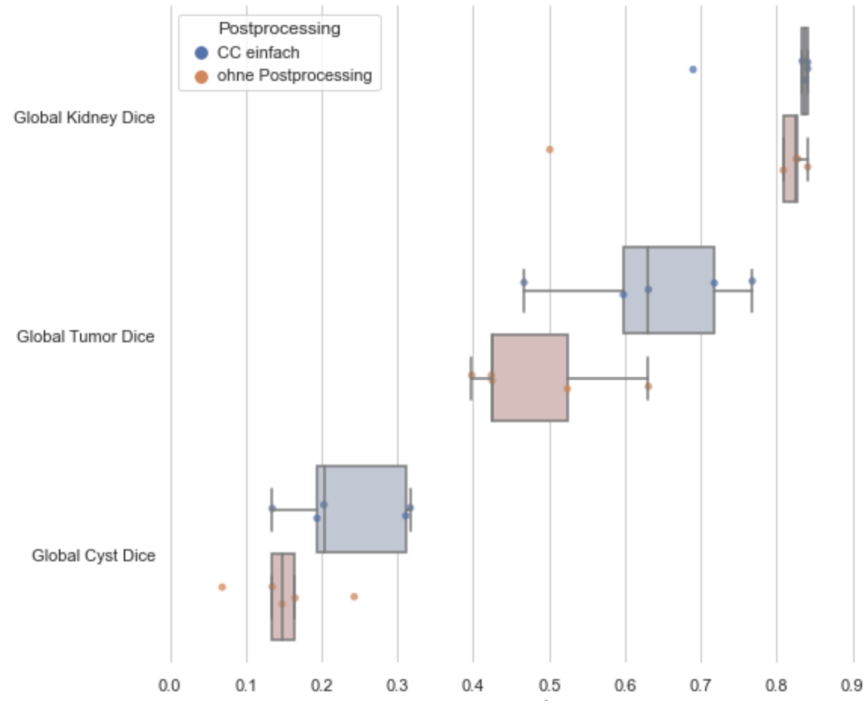


Abbildung 5.7: Vergleich der Dice Werte der einzelnen Klassen für Modelle mit Postprocessing CC einfach und ohne Postprocessing

Für die einzelnen Dice Werte der Klassen werden durch das Postprocessing ebenfalls Verbesserungen sichtbar. Besonders für den Tumor Dice verbessert sich der Median stärker als bei den anderen Klassen, aber auch die Streuung nimmt zu. Für den Kidney Dice fällt auf, dass durch die Nachverarbeitung die Ergebnisse der Modelle, bis auf einen Ausreißer, sehr nah beieinander liegen.

5.4.2 Connected Components Niere

Im Folgenden wird dargestellt, welche Veränderungen die beiden Postprocessing Ansätze bewirken. Betrachtet wird dabei die Methode *Connected Components* einfach, welche in Kapitel 4.5.3.1 (CC einfach) beschrieben wird, und *Connected Components* Niere aus Kapitel 4.5.3.2 (CC Niere). Es wurden insgesamt zehn Modelle ausgewählt. Dabei handelt es sich wieder um unterschiedliche Modelle bezüglich der Trainingsparameter. Jedes dieser Modelle wurde durch beide Postprocessing-Arten erweitert und ausgewertet.

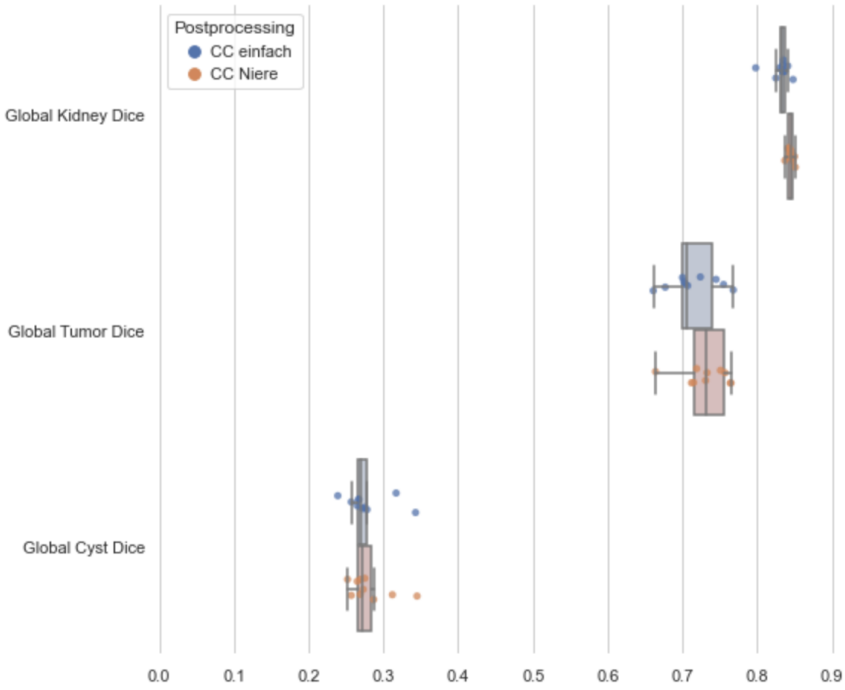


Abbildung 5.8: Vergleich der Dice Werte der einzelnen Klassen für Modelle mit Postprocessing CC Niere und mit Postprocessing CC einfach

Die größte Verbesserung lässt sich für die Klasse Tumor erkennen (Abbildung 5.8). Der Median für den Tumor Dice für die Modelle mit dem Postprocessing CC Niere liegt bei 0.732 und für CC einfach bei 0.706, während dieser Wert sich für Niere (CC Niere: 0.845, CC einfach: 0.834) und Zyste (CC Niere: 0.271, CC einfach: 0.27) weniger stark verändert. Allgemein sind die Verbesserungen weniger stark als im vorherigen Vergleich.

Um einen besseren Eindruck zu erhalten wie sich das Postprocessing auf Modelle auswirkt, folgt die Auswertung für die beiden Arten des Postprocessings für ein Modell. Die folgende Tabelle führt die detaillierten Ergebnisse für die verschiedenen Ansätze für die KiTS21 Metriken für ein Modell auf. Hierfür wurde ein 3D Modell mit 4 Leveln, $36 \times 36 \times 36$ Patchgröße und $1,5mm$ Voxelgröße ausgewertet. Die Werte für die weitere Konfiguration entsprechen den Defaultwerten, die im Kapitel 4.2.3 detailliert beschrieben werden.

Metrik	CC Niere	CC einfach
Tumor Dice	0.763	0.720
Tumor Surface Dice	0.589	0.562
Kidney and Masses Dice	0.956	0.914
Kidney and Masses Surface Dice	0.869	0.824
Kidney Mass Dice	0.799	0.757
Kidney Mass Surface Dice	0.637	0.606

Tabelle 5.14: Vergleich der KiTS21 Metriken für ein Modell mit Postprocessing CC Niere und mit Postprocessing CC einfach

Der Scatter Plot 5.9 zeigt die Ergebnisse pro Fall für den Tumor Dice für dieses Modell jeweils erweitert mit den beiden Postprocessing Schritten.

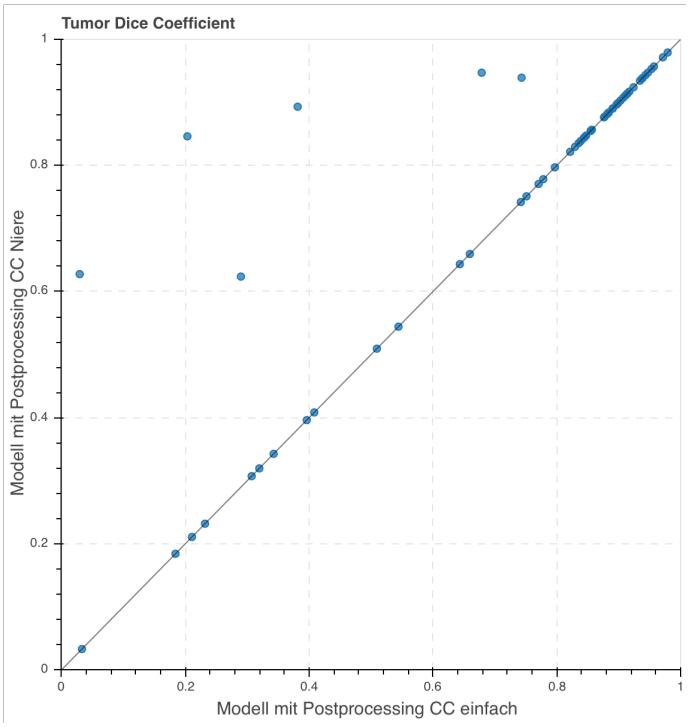


Abbildung 5.9: Vergleich des Tumor Dices pro Fall für ein Modell Modell mit Postprocessing CC Niere und mit Postprocessing CC einfach

Oberhalb der Geraden ist zu sehen, dass das Postprocessing CC Niere die Segmentierung in sechs Fällen verbessert und der Dice für die weiteren Fälle unverändert bleibt. Die Verbesserung des Tumor Dices, die in der Tabelle 5.14 abzulesen ist, basiert also auf der Verbesserung

für diese sechs Fälle. Für zwei dieser Fälle, 102 und 284, findet bei den meisten anderen Modellen durch das Postprocessing CC Niere ebenfalls eine Verbesserung statt, wie die Auswertung dieser ergab. Daher werden diese Fälle nun exemplarisch genauer betrachtet.

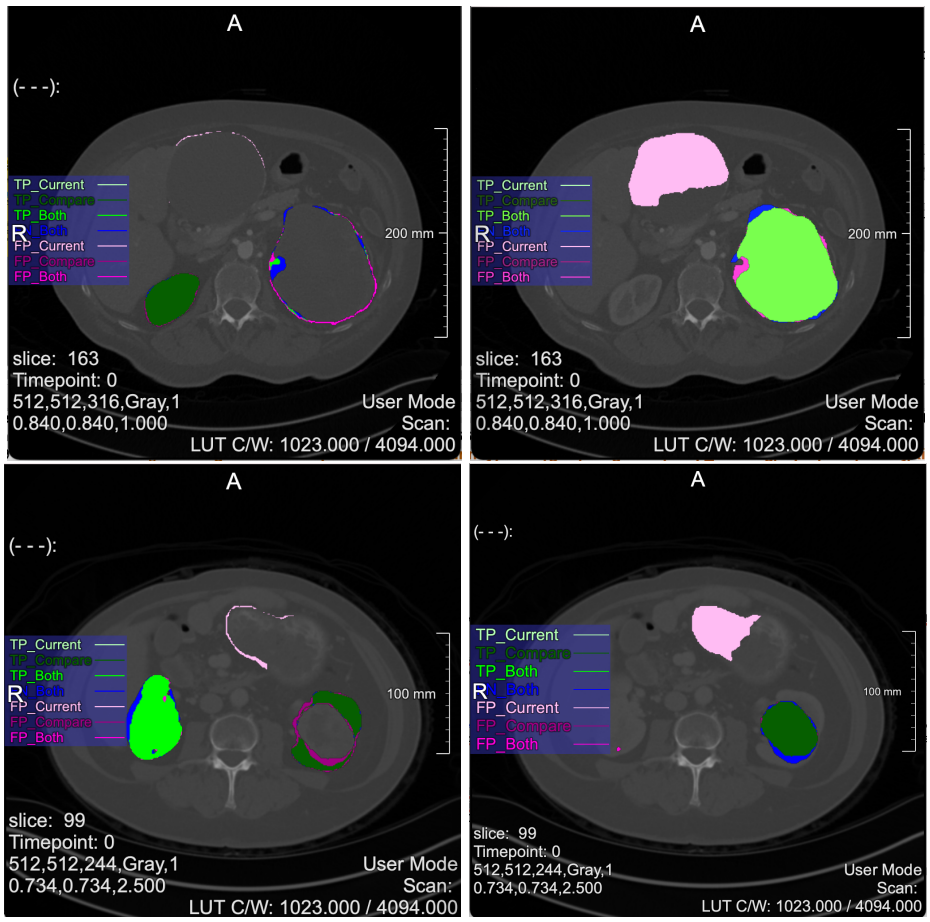


Abbildung 5.10: Current: Postprocessing CC einfach, Compare: Postprocessing CC Niere. Links: Segmentierung Niere. Rechts: Segmentierung Tumor. Oben: Fall 102. Unten: Fall 284.

Bei Fall 102 aus Abbildung 5.10 handelt es sich um den Fall, der bereits im Kapitel 4.5.3.2 vorgestellt wurde. Bei der Vorverarbeitung, bei der nur die zwei größten verbundenen Komponenten ausgewählt werden, wird hier die Leber fälschlicherweise als Tumor segmentiert und auf Grund des hohen Volumens anstelle der Niere ausgewählt - das verschlechtert den Tumor Dice. Ähnliches wird bei Fall 284 deutlich. Wieder wird fälschlicherweise die Leber, bzw. Teile der Leber, als Tumor segmentiert und dadurch wird eine der Nieren nicht als größte Komponente ausgewählt.

Hier verschlechtert nicht nur der falsch-positiv segmentierte Tumor das Ergebnis, sondern wurde auch durch die Nicht-Auswahl der Niere ein daran gebundene eigentlich richtig-positiv segmentierter Tumor entfernt. Durch das CC Niere Postprocessing wird nun der falsch-positiv segmentierte Tumor entfernt und die zweite Niere mit Tumor richtigerweise ausgewählt.
Beim direkten Vergleich zwischen *ohne Nachverarbeitung*, *Postprocessing mit CC einfach* und *Postprocessing CC Niere* zeigt sich, dass die Anwendung des einfachen Ansatzes bereits eine große Verbesserung bewirkt und die Erweiterung (CC Niere) diese zusätzlich verstärkt. Der Sprung ist jedoch geringer, siehe Abbildung 5.11.

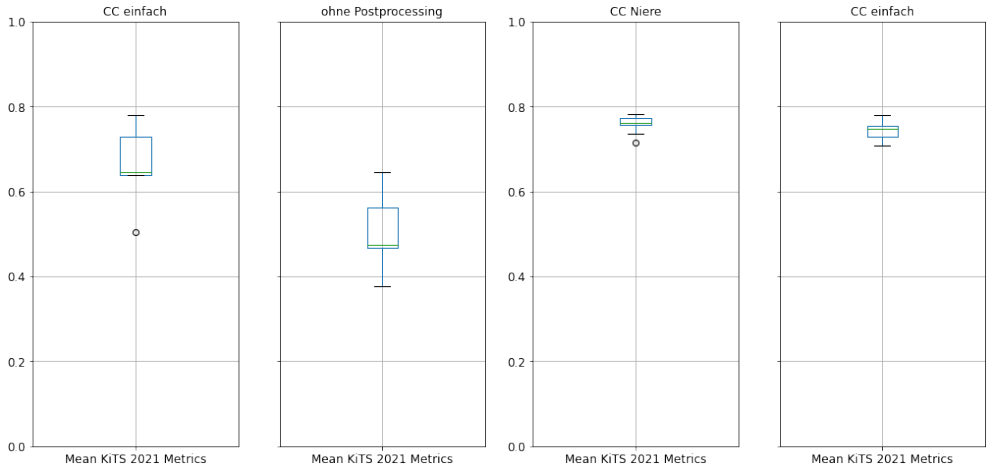


Abbildung 5.11: Vergleich der KiTS21 Mean Metrics ohne Postprocessing, mit Postprocessing CC einfach und CC Niere

Der Vergleich findet jeweils zwischen *ohne Nachverarbeitung* und *Postprocessing mit CC einfach* statt, basierend auf der Untersuchung von fünf Modellen mit und ohne Postprocessing. Der zweite Vergleich zeigt dann die Verbesserung zwischen *Postprocessing mit CC einfach* und *Postprocessing CC Niere*, dafür wurden zehn Modelle untersucht.

5.4.3 MajorityVote Läsion

Das in der Methode unter 4.5.3.3 beschriebene Postprocessing mit *MajorityVote* für Läsionen verbessert die Ergebnisse, wie die folgende Ta-

belle 5.15 zeigt. Als Modell wurde hierfür das der Submission für die KiTS21-Challenge ausgewählt. Es wird deutlich, dass der zweite Schritt, bei dem die Labels geändert werden, ausschließlich zu Verbesserungen von Tumor- und Tumor Surface Dice führen.

Metrik	Ohne PP	CC Niere	MajorityVote
Tumor Dice	0.524	0.765	0.781
Tumor Surface Dice	0.329	0.592	0.627
Kidney and Masses Dice	0.872	0.951	0.951
Kidney and Masses Surface Dice	0.722	0.904	0.904
Kidney Mass Dice	0.556	0.799	0.798
Kidney Mass Surface Dice	0.361	0.649	0.648

Tabelle 5.15: Vergleich der KiTS21 Metriken für das Submission Modell ohne Postprocessing (PP), mit Postprocessing CC Niere und mit Postprocessing MajorityVote

Mit Blick auf die Scatterplots 5.12 sieht man, dass es sowohl für die Tumor Dice pro Fall Werte als auch für die Dice Werte der Zyste in fast allen Fällen zu Veränderungen kommt, auch wenn diese teilweise minimal sind. Dies lässt sich damit erklären, dass es durch die Anpassung des Labels von Zyste auf Tumor oder andersherum immer auch zu einer Veränderung für die jeweils andere Klasse kommt. Außerdem kommt es in einigen wenigen Fällen nicht zu einer klaren Verbesserung, wie es bei dem Postprocessings CC Niere gegenüber Postprocessing CC einfach der Fall ist, sondern es lassen sich auch Verschlechterungen beobachten. Wie die Veränderung des Tumor Dices zeigt, überwiegen die Fälle, die durch eine Anpassung des Labels verbessert werden.

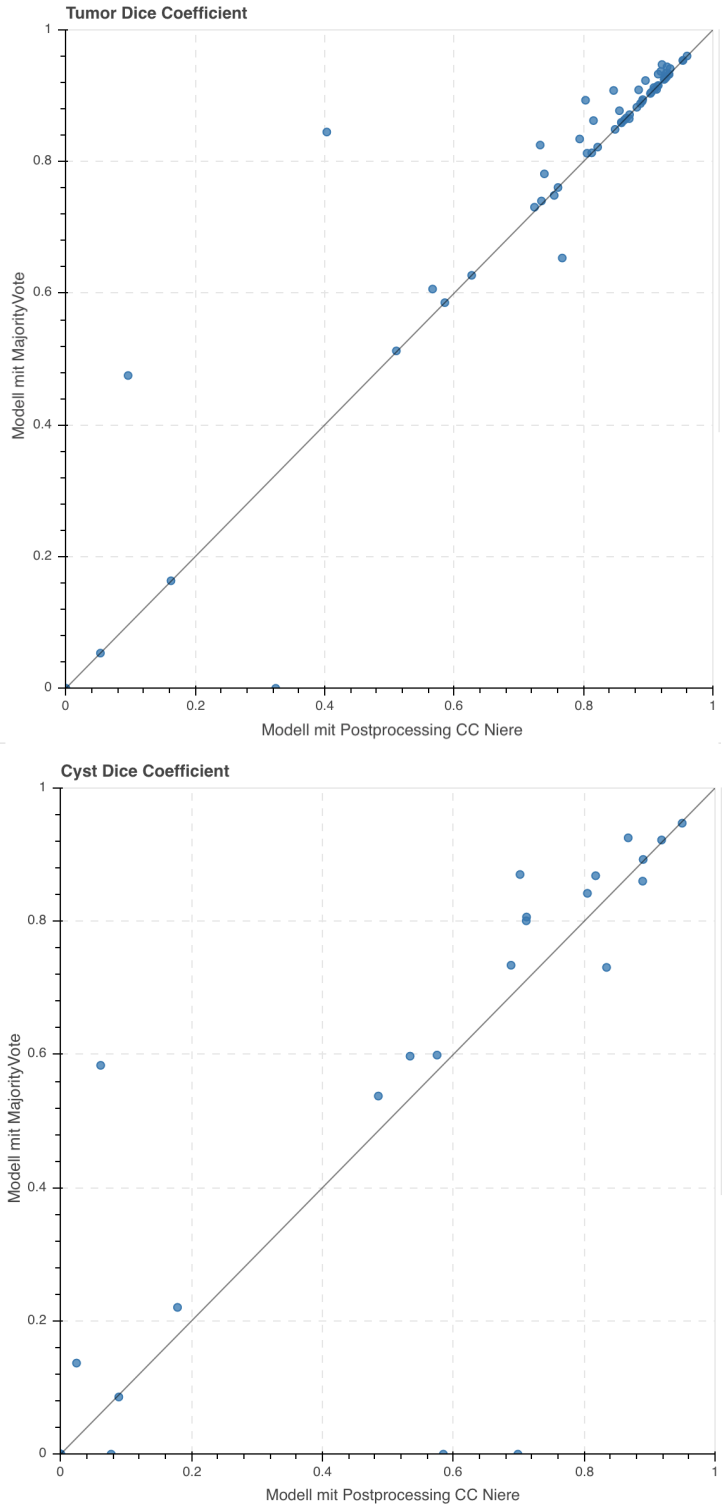


Abbildung 5.12: Vergleich des Tumor und Zysten Dice pro Fall für ein Modell mit Postprocessing CC Niere und mit Postprocessing CC einfach

5.4.4 Kaskade mit Klassifikator

AUTOR*IN: NIKLAS AGETHEN

Die Erweiterung des Segmentierers zu einer Kaskade mit Klassifikator versprach eine Verbesserung der Ergebnisse durch das einheitliche Segmentieren von Läsionen. Wie im Kapitel 4.5.3 beschrieben, wurden dazu verschiedene Versionen eines DNN-Klassifikators auf den Ausgaben eines Segmentierers, hier eines der U-ResNet-Modelle, trainiert. Wie dort beschrieben, nutzten wir zunächst den Lossplot und die Confusion-Matrix zum Vergleich der einzelnen Klassifikatoren. Beim Vergleich der Lossplots unterschiedlicher Klassifikatoren zeigte sich, dass die Netze mit Dropout und Data-Augmentation eine höhere Accuracy im Training erzielen. Da die Bedeutung von Dropout und Data-Augmentation speziell bei Trainings mit wenigen Trainingsdaten hoch einzuschätzen ist (siehe Kapitel 3.1), überraschen diese Beobachtungen nicht.

Die trainierten Modelle mit Dropout und Data-Augmentation erzielen mit etwa 80% (Bestwert) eine ordentliche Accuracy im Training, die sich auch entsprechend in der Confusion-Matrix (siehe Abbildung 5.13) widerspiegelt. Weitere Trainings mit unterschiedlichen Paddingformen und Filterzahlen unterscheiden sich im Lossplott nur geringfügig.

#	Kaskade	Postprocessing	Mean Metrics	Zysten Dice
1	Ja	CC Niere	0.718	0.000
2	Nein	CC Niere	0.730	0.300
3	Nein	CC Niere + Majority Vote	0.764	0.378

Tabelle 5.16: Vergleich eines Segmentierers mit und ohne Kaskade sowie alternativer Lösung über erweitertes Postprocessing

Die Auswertung des Gesamtmodells bestehend aus einem U-ResNet und dem besten Klassifikator zu diesem Segmentierer zeigte allerdings schlechtere Ergebnismetriken als das Modell ohne Kaskade (siehe Tabelle 5.16). Dabei wurden die KiTS21-Metriken für den selben Segmentierer mit und ohne Kaskade verglichen. In beiden Fällen kam zudem dasselbe Postprocessing zum Ausschluss der Artefakte zum Einsatz,

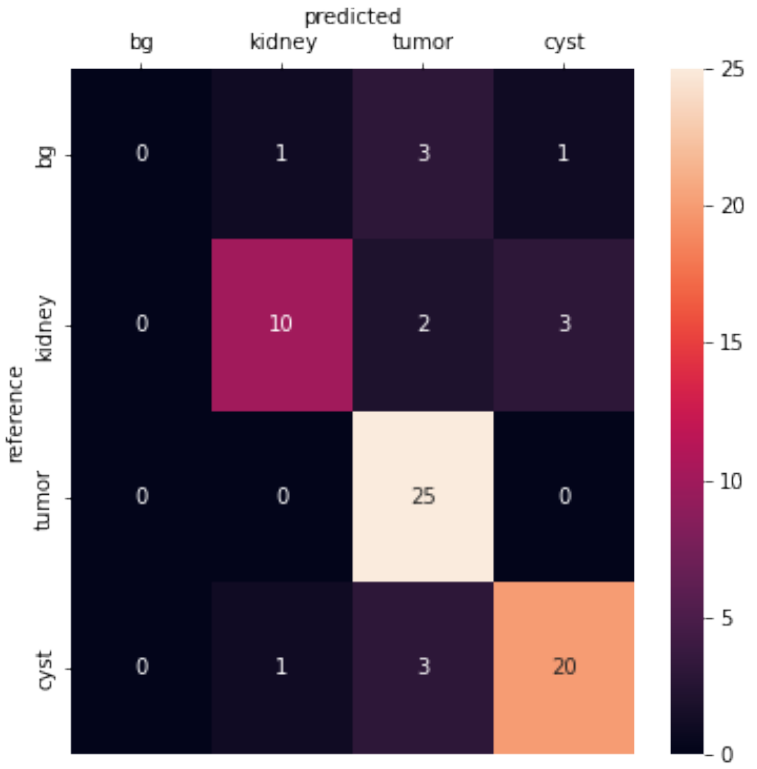


Abbildung 5.13: Confusion-Matrix einer CNN-Klassifikator-Kaskade

sodass sich die Modelle nur durch die Anwendung der Kaskade unterschieden. Speziell der sehr niedrige Zysten-Dice lässt an dieser Stelle einen Fehler in der Inferenz nicht ausschließen. Aufgrund der nahenden Frist der KiTS21-Challenge sowie dem Projektende wurden keine weitere Versuche unternommen, den Kaskadenansatz zu optimieren. Stattdessen wurde der Fehler uneinheitlicher Klassifikationen von Läsionen über das Postprocessing mit MajorityVote, wie im vorigen Unterkapitel beschrieben, adressiert. Die Tabelle 5.16 hebt zudem nochmal hervor, dass die Ergebnisse des ausgewählten Segmentierers durch dieses erweiterte Postprocessing verbessert werden konnten und daher eine bessere Alternative als der Ansatz der Kaskade darstellt.

6 Diskussion

6.1 KiTS21 Challenge

AUTOR*IN: LENA PHILIPP

Die Modelle, die für die Challenge trainiert wurden, basierten zum einen auf den Erkenntnissen, die aus der KiTS19 Challenge gewonnen wurden und zum anderen gab das nnU-Net weitere Impulse. Im Vergleich zu den Trainings mit den Daten der KiTS19 Challenge wurden mit den Daten der KiTS21 Challenge die Tumore um einiges besser erkannt trotz gleicher Konfiguration beim Training. Bei dem KiTS21 Datensatz handelt es sich um eine Erweiterung der vorhergehenden KiTS19 Daten, jedoch mit neuen Annotationen sowie der Zyste als zusätzliche Klasse. Das deutet daraufhin, dass die neuen Annotationen und zusätzlichen Daten zu einer Verbesserung der Netze bei der Erkennung von Tumoren geführt haben. Die wichtigste Erkenntnis, die wir aus dem nnU-Net gewonnen haben, ist, dass die Eigenschaften des Datensatzes essenziell für die Optimierung der Methodenkonfiguration sind. Dadurch sind Themen wie Voxelgröße und rezeptives Feld zentral geworden bei der Wahl von Architektureinstellungen. Dadurch, dass bei unseren Trainings Padding valid verwendet wurde, wie in der ursprünglichen U-Net Architektur, und beim nnU-Net Padding same, konnten die Erkenntnisse daraus nur bedingt übertragen werden.
Bei der KiTS21 Challenge stellte sich die Erkennung der oft kleinen Zysten, neben der Unterscheidung von Zysten und Tumoren, als größte Herausforderung heraus. Die wichtigste Voraussetzung für das Segmentieren der Zysten war die Verwendung des SPS beim Training - also das

Oversampling innerhalb der Batches. Für zukünftige Untersuchungen würde es sich anbieten weitere Sampling Kombinationen zu evaluieren um das Klassenungleichgewicht auszugleichen. Beim Training wurden hohe Jaccard Werte für die Zyste auf den Validierungsdaten angezeigt, jedoch konnten diese nicht auf unseren Testdaten erreicht werden. Eine Erklärung dafür könnte Overfitting sein. Für diesen Umstand wäre es interessant gewesen Kreuzvalidierung anzuwenden um dies genauer zu untersuchen. Außerdem hätte die Validierungsstrategie angepasst werden können. Verbesserungen wurden außerdem durch die Verwendung von Data Augmentation beobachtet. Wieder profitierten davon besonders die seltenere Klasse Zyste und auch für die Tumore zeigen sich Verbesserungen. Eine Erklärung dafür kann darin liegen, dass die Formen und Variationen diverser sind als bei den Nieren, wodurch die Erweiterung der Trainingsdaten durch leicht veränderte Varianten dieser Klassen die Generalisierungsleistung verbessert. Das Anpassen der Voxelgröße auf Werte zwischen $1mm$ und $1,5mm$ führte für die Klassen Tumor und Zyste zu Verbesserungen, dies scheinen Werte zu sein, die zu einem guten Kompromiss zwischen detaillierten und kontextbezogenen Informationen führen. Bei den Nieren zeigen sich keine großen Veränderungen für die Voxelgrößen $1mm$ bis $2,5mm$. Eine überraschende Beobachtung war, dass das Hinzufügen einer weiteren Schicht und damit der Vergrößerung des rezeptiven Feldes, nicht zu einer Verbesserung der Ergebnisse führte. Eine Erklärung hierfür könnte sein, dass das rezeptive Feld bereits ausreichend groß war.

Neben der Vorverarbeitung und der Trainingskonfiguration wurden wesentliche Verbesserungen durch Postprocessing mit Connected Components und MajorityVote erzielt, das bei der finalen Abgabe zum Einsatz kam. Das Thema Postprocessing ist erst im späteren Verlauf des Projektes aufgekommen, sodass einige der Ideen diesbezüglich aus zeitlichen Gründen nicht rechtzeitig zufriedenstellend umgesetzt werden konnten, wie die nachträgliche Klassifikation durch ein neuronales Netz. Weitere Themen wie Transfer Learning und Data Augmentation mit GANs konnten ebenfalls aus zeitlichen Gründen nicht vertieft und angewendet werden.

6.2 Challengr

AUTOR*IN: JANNES ADAM

Challengr wurde für uns im Laufe des Projekts ein immer wichtigeres Werkzeug zur Bewertung von Sessions, zum Speichern von deren Parametern und damit zur Parameterauswahl beim Design neuer Sessions. Durch das Verwenden wurde die Entwicklung vorangetrieben: Wo Möglichkeiten zum Vergleich von Sessions fehlten, wurden welche geschaffen und wo die Verwendung umständlich war oder Fehler gefunden wurden, wurde an der Verbesserung gearbeitet.
Ein Fehler, der schon zu Beginn häufig auftrat, war ein Timeout beim Laden der Sessions. Zuerst bestand der Verdacht, dass es an zu großen Datenmengen läge, die zeitgleich geladen werden müssen. Daher entstand der Plan die Sessions iterativ zu laden. Die implementierte Lösung führte jedoch dazu, dass für das Laden im Regelfall etwa die zehnfache Zeit benötigt wurde. Der Timeout trat trotz des iterativen Ladens jedoch trotzdem noch auf. Da der Fehler nicht lokalisiert werden konnte, konnte zunächst keine Lösung dafür implementiert werden.
Im Laufe des Projekts wurde an vielen Stellen die Usability verbessert, indem die Übersichtlichkeit wie bei der Case Data Table verbessert wurde oder einige Einstellungen gespeichert wurden, die die Individualisierung von Challengr verbessern.
Für die Verwendung neuer Features wurden HTML-Tabellen in Quasar-Tabellen konvertiert. Dadurch konnten Möglichkeiten für neue Features, wie den Ladeindikator oder die Suche, geschaffen werden. Zusätzlich hatte das den Vorteil, dass der Code kompakter und einfacher wartbar wurde.
Die größte Veränderung, die in diesem Projekt an Challengr geschaffen wurde, ist der Compare-Multiple-Sessions-Tab, der ganz neu erdacht und implementiert wurde. Er erlaubt den Vergleich von mehr als zwei, bzw. allen Sessions, der vorher noch nicht möglich war. Dazu wurde auch ein Diagramm entwickelt, welches die Verteilung der Evaluationsmetriken der einzelnen Cases für die ausgewählten Sessions anzeigt.
Für diesen Tab wurde ursprünglich eine Scatterplot-Matrix zur Untersu-

chung des Einflusses mehrerer Parameter auf die Ergebnisse geplant, der jedoch aus Zeitgründen nicht umgesetzt werden konnte.
Insgesamt wurde Challengr, auch dank der neuen Features und der verbesserten Usability, viel innerhalb des Projektes genutzt und erwies sich als sehr hilfreich.

7 Fazit

AUTOR*IN: RENE FINZEL

Im Masterprojekt DeepAnatomy ging es um die Entwicklung und Implementierung von Bildsegmentierungsalgorithmen, die Visualisierung der Ergebnisse über die Plattform Challengr und abschließend um die Teilnahme bei der KiTS21-Challenge.
Eine besondere Herausforderung hierbei war, dass das gesamte Semester - aufgrund der Corona-Pandemie - komplett online stattfand. Dadurch war es schwieriger, sich als Team kennenzulernen und produktiv miteinander zu arbeiten. Eine große Hilfe hierbei war der Discordserver. Über Discord fand die komplette Organisation außerhalb des Plenums statt, die Teams hatten die Möglichkeit Meetings abzuhalten und die Student*innen konnten mithilfe des Sprachchats und Bildschirmübertragung gemeinsam an einer Aufgabe arbeiten und sich gegenseitig unterstützen. Zusätzlich fanden in regelmäßigen Abständen Spieleabende statt, die dabei geholfen haben, sich untereinander besser kennenzulernen.
Durch die Möglichkeit sich selbst Ziele für das Projekt zu stecken, war es den Student*innen möglich, eines der vorgegebenen Themen auszusuchen und sich auf diesem Gebiet zu spezialisieren. Dadurch haben sich verschiedene Teams gebildet, die nicht nur geschlossen für sich arbeiteten, sondern auch teamübergreifend kommunizieren mussten, um die Anderen mit den Ergebnissen der eigenen Arbeit zu unterstützen.
Das Challengr-Team hat durch Bugfixes, Anpassungen und Verbesserungen an der bestehenden Challengr-Software erreicht, dass sich die Usability der Software verbessert hat und sie einen übersichtlicheren Vergleich zwischen den zwei gewählten Netzen bietet. Zusätzlich kam mit der Möglichkeit, die Parameter und Evaluationsmaße von mehr als nur

zwei Netze auf einmal zu vergleichen, auch eine komplett neue Funktionalität hinzu, die es einfacher macht, die Resultate von mehr als zwei verschiedenen Modellen gleichzeitig zu vergleichen.
Neben der Arbeit an Challengr, haben Student*innen Experimentenserien mit der Deep Medic-, Attention-U-Net- und U-ResNet-Architektur auf den Daten der KiTS19-Challenge durchgeführt. Dabei wurden z.B. Trainings der einzelnen Modelle mit verschiedenen Parametern durchgeführt und geschaut, welche die besten Ergebnisse liefern. Zusätzlich wurde getestet wie sich z.B. Oversampling, Postprocessing oder Data Augmentation auf die Resultate der Experimente auswirken.
Durch die gewonnene Erfahrung aus den Experimentenserien, den Verbesserungen und Erweiterungen an Challengr und der Unterstützung der Betreuer*innen war es möglich, gegen Ende des Projekts an der KiTS21-Challenge teilzunehmen.
Unsere Ergebnisse für die KiTS21-Challenge haben gezeigt, dass allein eine Erweiterung der KiTS19-Trainingsdaten durch neue Annotationen und der Zyste als weitere Klasse zu einer Verbesserung der Ergebnisse geführt haben. Das zeigt, dass die gründliche Aufbereitung von Trainingsdaten ein wichtiger Schritt ist und auch so schon bessere Ergebnisse erzielt werden können, ohne die Modelle anpassen zu müssen.
Die Unterscheidung von Zysten und Tumoren sowie die Erkennung von kleinen Zysten stellte sich trotzdem als problematisch dar. Durch die Verwendung von Oversampling, Data Augmentation und Anpassung der Voxelgrößen, wurden allerdings signifikante Verbesserungen der Ergebnisse beobachtet. Abschließend wurde das optimierte Modell zusammen mit einem Paper für die KiTS21-Challenge eingereicht. Das Paper, sowie das Modell, wurden akzeptiert. Innerhalb der Challenge belegte das Modell den 12. Platz. Das Paper wurde auf der MICCAI im Rahmen der Satellite Events vorgestellt. Außerdem wird das Paper voraussichtlich in den Proceedings zu dieser Veranstaltung erscheinen. Zusätzlich gab es die Möglichkeit an einer abschließenden Umfrage teilzunehmen, die innerhalb einer weiteren Veröffentlichung ausgewertet wird und die Teilnehmer*innen als Co-Autor*innen aufnimmt.

8 Ausblick

AUTOR*IN: LENA PHILIPP

Mit Blick auf das übergeordnete Ziel des Projektes eine einfach bedienbare Plattform zu entwickeln, würde es sich anbieten die bisherigen Entwicklungen mit externen Personen zu evaluieren. So wäre eine mögliche Fragestellung, wie hilfreich die Erweiterungen von Challengr für Projektfremde Dritte sind. Als konkretes Thema für die Erweiterung von Challengr bietet sich Visualisierung der Modellergebnisse mit Blick auf unterschiedliche Parameter an. Besonders für Personen, die ohne tieferes Fachwissen aus dem Bereich Deep Learning mit dem Tool arbeiten, wäre dies möglicherweise ein Weg, einen leichteren Zugang zu dem komplexen Thema des Zusammenspiels der Hyperparameter zu finden.

Des Weiteren zeigte sich während des Projekts, wie wichtig es für die Konfiguration von Trainings ist, die Daten, mit denen gearbeitet wird, zu verstehen und zu analysieren. Da es sich bei medizinischer Bildverarbeitung um ein interdisziplinäres Feld handelt, bieten sich Möglichkeiten zur Kooperation mit Mediziner*innen und / oder Radiolog*innen an, um einen Wissenstransfer zu ermöglichen. Die Meinung von Personen dieser Gruppen wären außerdem eine gute Unterstützung, um weitere Impulse für die Weiterentwicklung der angedachten Plattform zu erhalten, da sie eine mögliche Benutzer*innenzielgruppe sein könnten. Auch wurde deutlich, dass die Themen Datenanalyse und Nachverarbeitung wichtig für die Optimierung des Netzes sind. Bezüglich der Datenanalyse wäre es lohnenswert, über mögliche Automatisierungen nachzudenken beispielsweise durch das Entwickeln eines separaten Tools.
Für zukünftige Projekte könnte es sich außerdem als hilfreich herausstellen, schon an früheren Zeitpunkten im Projekt über die Nachverarbeitung

von Ergebnissen des Netzes zu sprechen und zu beobachten, wie und ob sich der Umgang mit Daten und neuronalen Netzen als „Black Box“ verändert.

Abkürzungsverzeichnis

Adam Adaptive Moment Estimation

AG Attention Gate

BS Batchsize

CNN Convolutional Neural Network

CT Computertomographie

DNN Deep Neural Network

FCN Fully Convolutional Network

FN False negative

FP False positive

GAN Generative Adversarial Network

HEC Hierarchical Evaluation Class

HU Hounsfield Unit

KiTS Kidney Tumor Segmentation

MRT Magnetresonanztomographie

ReLU Rectified Linear Unit

RPC Remote Procedure Call

SPS Stratified Patch Sampler

TN True negative

TP True positive

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] https://kits21.kits-challenge.org/, 2021.

[2] MeVis Medical Solutions AG. https://www.mevislab.de/, 2021.

[3] Fraunhofer MEVIS. Redleaf dokumentation. RedLeaf Source Code, 2018.

[4] Li Deng and Dong Yu. Deep learning: Methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

[5] Phil Wennker. *Machine Learning*, pages 9–37. Springer Fachmedien Wiesbaden, Wiesbaden, 2020.

[6] Sharma Sagar. Activation functions in neural networks. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6, 09 2017.

[7] Haoning Lin, Zhenwei Shi, and Zhengxia Zou. Maritime semantic labeling of optical remote sensing images with multi-scale fully convolutional network. *Remote Sensing*, 9:480, 05 2017.

[8] cs231n. Convolutional neural networks. https://cs231n.github.io/convolutional-networks/.

[9] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights Imaging*, 9:611–629, 2018.

[10] Gerhard Paaß and Dirk Hecker. *Bilderkennung mit tiefen neuronalen Netzen*, pages 119–166. Springer Fachmedien Wiesbaden, Wiesbaden, 2020.

[11] Vincent Andrearczyk and Paul F. Whelan. Chapter 4 - deep learning in texture analysis and its application to tissue image classification. In Adrien Depeursinge, Omar S. Al-Kadi, and J.Ross Mitchell, editors, *Biomedical Texture Analysis*, The Elsevier and MICCAI Society Book Series, pages 95–129. Academic Press, 2017.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[13] Arc. Convolutional neural network. https://towardsdatascience.com/convolutional-neural-network-17fb77e76c05, 12 2018.

[14] Andreas Folkers. *Grundlagen des Deep Learning*, pages 3–13. Springer Fachmedien Wiesbaden, Wiesbaden, 2019.

[15] Martin Werner. *Neuronale Netze mit Faltungsschichten*, pages 409–465. Springer Fachmedien Wiesbaden, Wiesbaden, 2021.

[16] Jiwon Jeong. The most intuitive and easiest guide for convolutional neural network. https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480, 01 2019.

[17] Andreas Moring. *Künstliche Intelligenz*, pages 25–41. Springer Berlin Heidelberg, Berlin, Heidelberg, 2021.

[18] Shruti Jadon. A survey of loss functions for semantic segmentation, 10 2020.

[19] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. *2016 Fourth International Conference on 3D Vision (3DV)*, pages 565–571, 2016.

[20] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[21] Ishan Shrivastava. Handling class imbalance by introducing sample weighting in the loss function.

[22] Yoshua Bengio. *Practical Recommendations for Gradient-Based Training of Deep Architectures*, pages 437–478. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[23] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[24] Prakhar. Intuition of adam optimizer. https://www.geeksforgeeks.org/intuition-of-adam-optimizer/, 10 2020.

[25] Vitaly Bushaev. Adam — latest trends in deep learning optimization. https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c, 10 2018.

[26] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[27] Alex Hernandez-Garcia and Peter König. Data augmentation instead of explicit regularization. 06 2018.

[28] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6:1–48, 2019.

[29] Hoo-Chang Shin, Neil A. Tenenholtz, Jameson K. Rogers, Christopher G. Schwarz, Matthew L. Senjem, Jeffrey L. Gunter, Katherine P. Andriole, and Mark Michalski. Medical image synthesis for data augmentation and anonymization using generative adversarial networks. In Ali Gooya, Orcun Goksel, Ipek Oguz, and Ninon Burgos, editors, *Simulation and Synthesis in Medical Imaging*, pages 1–11, Cham, 2018. Springer International Publishing.

[30] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[31] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[32] Lutz Prechelt. *Early Stopping — But When?*, pages 53–67. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[33] `http://cnx.org/content/col11496/1.6/`, 2013. CC BY 4.0.

[34] Prof. Dr. med. Rolf W. Günther Prof. Dr. med. Andreas H. Mahnken. Skript radiologie - grundlagen der diagnostik und intervention, 2013. `https://www.ukgm.de/ugm_2/deu/umr_rdi/Radio-Skript_UMR_1_02.pdf`, 2021-09-29.

[35] François Chollet. *Deep Learning with Python*. Manning Publications Co., 20 Baldwin Road, Shelter Island, NY 11964, 2018.

[36] Neeraj Dhungel, Gustavo Carneiro, and Andrew P. Bradley. Automated mass detection in mammograms using cascaded deep learning and random forests. In *2015 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–8, 2015.

[37] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, October 2001.

[38] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.

[39] Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven Mcdonagh, Nils Y Hammerla, Bernhard Kainz, Ben Glocker, and Daniel Rueckert. Attention U-Net: Learning Where to Look for the Pancreas. Technical report, 2018.

[40] Jo Schlemper, Ozan Oktay, Michiel Schaap, Mattias Heinrich, Bernhard Kainz, Ben Glocker, and Daniel Rueckert. Attention Gated Networks: Learning to Leverage Salient Regions in Medical Images. Technical report, 2019.

[41] L. A. Steiner. Adaptive control processes.richard bellman. oxford university press, princeton university press. 1961. 255 pp. 21 figures. 42s. *The Journal of the Royal Aeronautical Society*, 66(619):466–467, 1962.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[43] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[44] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[45] Nikolaos Zioulis, Antonis Karakottas, Dimitrios Zarpalas, and Petros Daras. Omnidepth: Dense depth estimation for indoors spherical panoramas. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 453–471, Cham, 2018. Springer International Publishing.

[46] Konstantinos Kamnitsas, Christian Ledig, Virginia F.J. Newcombe, Joanna P. Simpson, Andrew D. Kane, David K. Menon, Daniel Rueckert, and Ben Glocker. Efficient multi-scale 3d cnn with fully connected crf for accurate brain lesion segmentation. *Medical Image Analysis*, 36:61–78, 2017.

[47] Konstantinos Kamnitsas, Enzo Ferrante, Sarah Parisot, Christian Ledig, Aditya V. Nori, Antonio Criminisi, Daniel Rueckert, and Ben Glocker. Deepmedic for brain tumor segmentation. In Alessandro Crimi, Bjoern Menze, Oskar Maier, Mauricio Reyes, Stefan Winzeck, and Heinz Handels, editors, *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, pages 138–149, Cham, 2016. Springer International Publishing.

[48] Deep medic implementierung. `https://github.com/deepmedic/deepmedic`, 2021.

[49] F. Isensee, P.F. Jaeger, S.A.A. Kohl, J. Petersen, and K.H. Maier-Hein. nnu-net: a self-configuring method for deep learning-based biomedical image segmentation. *Nat Methods*, 18:203–211, 02 2021.

[50] Fabian Isensee. *From Manual to Automated Design of Biomedical Semantic Segmentation Methods*. PhD thesis, 2020.

[51] Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006. ROC Analysis in Pattern Recognition.

[52] Harald Peter. Dna-microarrays zur parallelen detektion und genotypisierung toxischer cyanobakterien, 2009.

[53] Iwan Binanto, Harco Leslie Hendric Spits Warnars, Bahtiar Saleh Abbas, Yaya Heryadi, Nesti Fronika Sianipar, Lukas, and Horacio Emilio Perez Sanchez. Comparison of similarity coefficients on morphological rodent tuber. In *2018 Indonesian Association for Pattern Recognition International Conference (INAPR)*, pages 104–107, 2018.

[54] Valerio Arnaboldi, Andrea Passarella, Marco Conti, and Robin I.M. Dunbar. Chapter 5 - evolutionary dynamics in twitter ego networks. In Valerio Arnaboldi, Andrea Passarella, Marco Conti, and Robin I.M. Dunbar, editors, *Online Social Networks*, Computer Science Reviews and Trends, pages 75–92. Elsevier, Boston, 2015.

[55] Stanislav Nikolov, Sam Blackwell, Alexei Zverovitch, Ruheena Mendes, Michelle Livne, Jeffrey De Fauw, Yojan Patel, Clemens Meyer, Harry Askham, Bernardino Romera-Paredes, Christopher Kelly, Alan Karthikesalingam, Carlton Chu, Dawn Carnell, Cheng Boon, Derek D'Souza, Syed Ali Moinuddin, Bethany Garie, Yasmin McQuinlan, Sarah Ireland, Kiarna Hampton, Krystle Fuller, Hugh Montgomery, Geraint Rees, Mustafa Suleyman, Trevor Back, Cían Hughes, Joseph R. Ledsam, and Olaf Ronneberger. Deep learning to achieve clinically applicable segmentation of head and neck anatomy for radiotherapy, 2021.

[56] Joshua V Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A Saurous. Tensorflow distributions. *arXiv preprint arXiv:1711.10604*, 2017.

[57] Nikhil Ketkar. Introduction to keras. In *Deep learning with Python*, pages 97–111. Springer, 2017.

[58] Tensorflow novograd documentation. `https://www.tensorflow.org/addons/api_docs/python/tfa/optimizers/NovoGrad` , Zugriffsdatum: 2021-09-29.

[59] Boris Ginsburg, Patrice Castonguay, Oleksii Hrinchuk, Oleksii Kuchaiev, Vitaly Lavrukhin, Ryan Leary, Jason Li, Huyen Nguyen, Yang Zhang, and Jonathan M. Cohen. Stochastic gradient methods with layer-wise adaptive moments for training of deep networks. 2020.