Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Master Künstliche Intelligenz

**Studienarbeit**

von

Helge **Kohl**

# Deep Vision

Medical Segmentation: nnUNet vs ResSegNet

# Inhaltsverzeichnis

**Abbildungsverzeichnis** **ii**

**1 Einleitung** **1**

**2 Materials** **2**
2.1 Allgemeines . . . . . 2
2.2 Verwendung des Datensatzes . . . . . 3

**3 Methoden** **4**
3.1 nnU-Net . . . . . 4
3.1.1 Allgemein . . . . . 4
3.1.2 Anwendung . . . . . 5
3.2 Monai SegResNet . . . . . 7
3.2.1 Allgemein . . . . . 7
3.2.2 Anwendung . . . . . 8

**4 Ergebnisse und Diskussion** **13**

**Literaturverzeichnis** **16**

# Abbildungsverzeichnis

2.1 Beispiel Bilddaten . . . . . 2
2.2 Beispiel Label (1 Channel pro Label) . . . . . 3

3.1 nnU-Net automatisierte Konfiguration Quelle: Isensee et al., 2020, S. 3 . 5
3.2 Schematische Visualiserung der Netzwerkarchitektur von SegResNet Quelle: Myronenko, 2018, S. 3 . . . . . 7
3.3 Visualisierung der Trainingsgeschwindigkeit mit verschiedenen Dataloadern Quelle: „PersistentDataset, CacheDataset, and simple Dataset Tutorial and Speed Test“, 2020 . . . . . 8
3.4 Bild vor und nach dem RandSpatialCrop . . . . . 9
3.5 Bild und Label 1 vor dem Flip . . . . . 9
3.6 Bild und Label 1 nach dem Flip auf Achse 0 . . . . . 10
3.7 Bild und Label 1 nach dem Flip auf Achse 1 . . . . . 10
3.8 Bild und Label 1 nach dem Flip auf Achse 2 . . . . . 11
3.9 Intensitätstransformationen . . . . . 11

4.1 Ergebnis nnU-Net . . . . . 14
4.2 Ergebnis SegResNet . . . . . 14
4.3 Trainingsverlauf SegResNet . . . . . 15

# Kapitel 1

# Einleitung

In dieser Arbeit geht es um die Segmentierung von medizinischen Bildern. Dabei soll das Framework nnUNet mit dem SegResNet des MONAI Frameworks verglichen werden. In den folgenden Kapiteln soll zuerst auf den verwendeten Datensatz eingegangen werden. Danach soll die Anwendung der beiden Architekturen erläutert werden. Zuletzt werden die Ergebnisse näher beleuchtet und evaluiert.

# Kapitel 2

# Materials

## 2.1 Allgemeines

Als Datensatz wurde der BrainTumour-Datensatz der Medical Segmentation Decathlon Challenge von 2018 (Antonelli et al., 2021) verwendet. Dieser besteht aus 750 MRT-Scans von Patienten die entweder mit einem Glioblastom oder einem niedriggradiges Gliom, also einem Hirntumor, diagnostiziert wurden. Die Bilder enthalten vier MRT-Sequenzen. Die Daten stammen von 19 verschiedenen Institutionen und wurden mit unterschiedlichem Equipment und Aufnahmeprotokollen aufgezeichnet. Die Scans wurden durch Experten der Neuroradiologie mit ihren Labels versehen.

| Label | Bezeichnung |
|---|---|
| 0 | Background |
| 1 | Edema |
| 2 | non-enhancing tumour |
| 3 | enhancing tumour |

**Tabelle 2.1:** Label des Datensatzes

Die Bilder haben eine Höhe und Breite von 240 Pixeln und eine Tiefe von 155. Durch die 4 Modalitäten hat ein Datensatz also die Form 4x240x240x155. Bis auf die Modalitäten gilt dasselbe für die Label, welche daher eine Form von 1x240x240x155 haben.

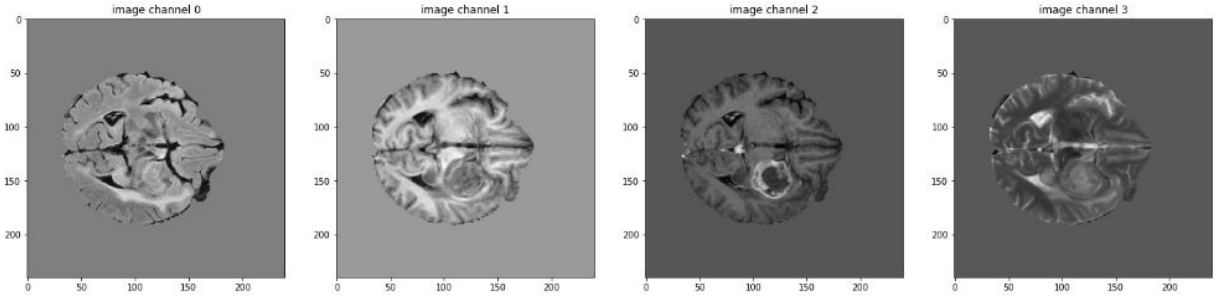


**Abbildung 2.1:** Beispiel Bilddaten

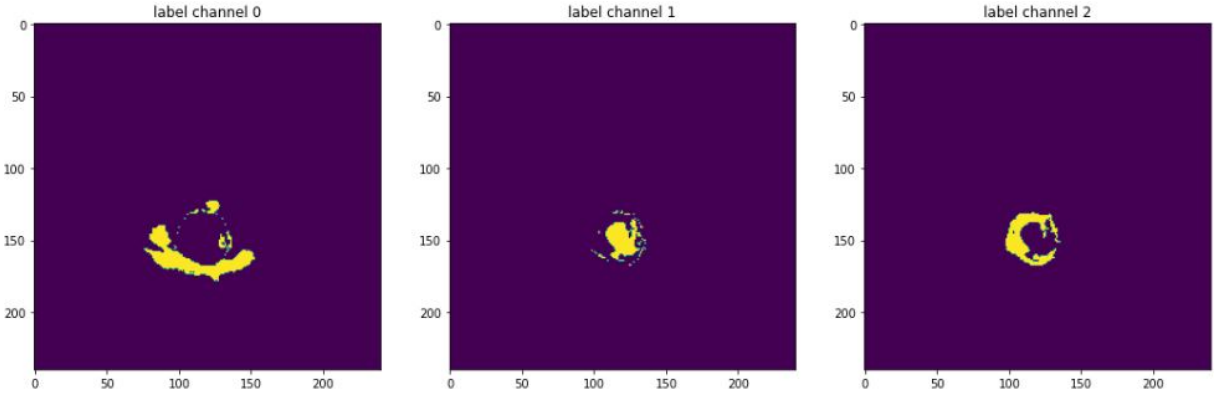


**Abbildung 2.2:** Beispiel Label (1 Channel pro Label)

## 2.2 Verwendung des Datensatzes

Die Daten im originalen Datensatz sind folgendermaßen aufgeteilt:

- Anzahl Trainingsdaten: 484
- Anzahl Testdaten: 266

Um die Daten am Ende jedoch sinnvoll evaluieren zu können, werden auch für die Testdaten Label benötigt. Diese liegen im Original jedoch nicht vor. Deshalb wurde der Trainingsdatensatz, in dem Label enthalten sind, folgendermaßen aufgeteilt:

- Anzahl Trainingsdaten: 338 (70%)
- Anzahl Validierungsdaten: 97 (20%)
- Anzahl Testdaten: 49 (10%)

# Kapitel 3

# Methoden

## 3.1 nnU-Net

### 3.1.1 Allgemein

nnU-Net von Isensee et al. steht für „no new (U-)Net“ und ist ein Framework zur Segmentierung von medizinischen Bildern basierend auf der U-Net-Architektur. Das besondere an nnU-Net ist, dass es nur mit einem Set von drei U-Net-Modellen mit kleinen Änderungen zum original U-Net-Modell arbeitet. Es wurde speziell dafür Entworfen auf nahezu allen medizinischen Datensätzen zu funktionieren ohne es für einen bestimmten Anwendungsfall anzupassen. nnU-Net konfiguriert sich automatisiert selbst durch Adaption am Datensatz und konzentriert sich dabei auf Schritte bei denen viel Performanz gewonnen werden kann, wie:

- Preprocessing (z.B. Resampling und Normalisierung)
- Training (z.B. Loss- und Optimizereinstellungen sowie Data-Augmentation)
- Inferenz (z.B. Patch basierte Strategie und Ensembling)
- Post-Processing (z.B. erzwingen von Single-Connected Components)

Wie in Abbildung 3.1 zu sehen ist basiert die automatische Konfiguration auf drei Parametergruppen: fixierte, regelbasierte und empirische Parameter. Dabei werden zuerst die festen Parameter gesammelt und optimiert. Im zweiten Schritt werden die regelbasierten Parameter ermittelt. Hierfür werden Metadaten des Datensatzes, wie Bildgröße und Modalität, herangezogen aber auch andere Faktoren wie der verfügbare GPU-Speicher werden beachtet. Anhand dieser Daten werden die drei U-Net-Modelle trainiert. Im dritten Schritt werden die Trainingsdaten ausgewertet und abhängig davon wird das Modell sowie die Post-Processing-Routine festgelegt. Die Art und Weise wie nnU-Net arbeitet wurde an zehn verschiedenen Datensätzen der Medical Decathlon Segmentation Challenge getestet und erreichte den höchsten Dice-Score für alle Klassen, außer Klasse 1 BrainTumour, der Phase 1 Aufgaben. (Isensee et al., 2018)

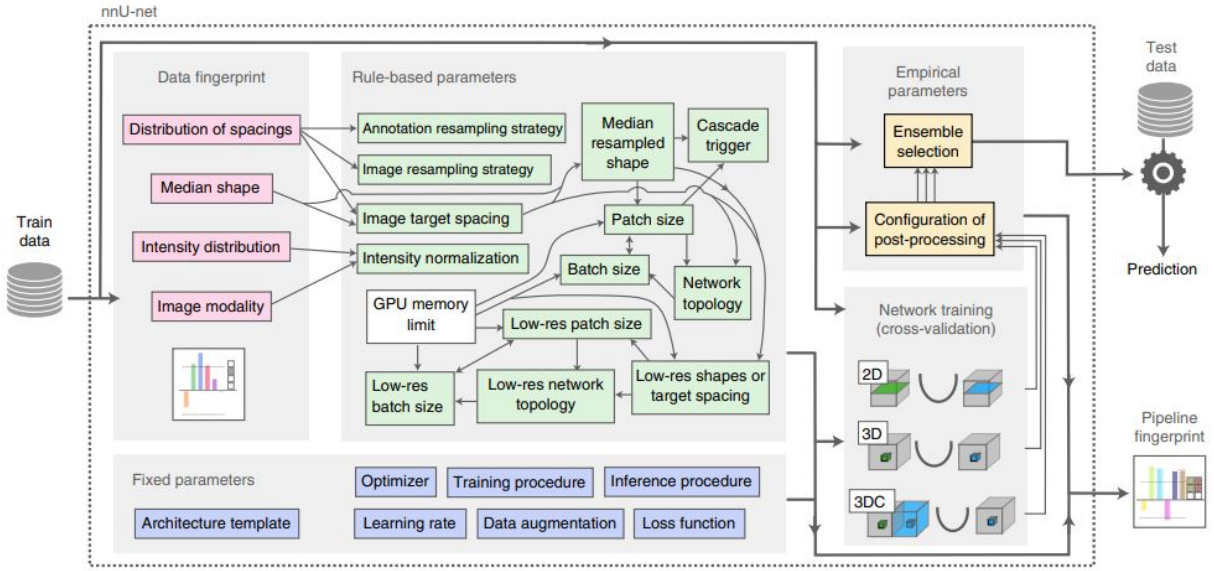


**Abbildung 3.1:** nnU-Net automatisierte Konfiguration Quelle: Isensee et al., 2020, S. 3

### 3.1.2 Anwendung

Im Folgenden gehe ich nun auf die Anwendung von nnU-Net in der Studienarbeit ein. Um die Arbeitsschritte besser nachvollziehen zu können wurden diese in einem Jupyter Notebook zusammengefasst. Dieses wurde auf Basis des nnU-Net Workshops von Lüth et al., 2022 erstellt. In diesem wurden folgende Arbeitsschritte durchgeführt:

- Aufsetzen der nnU-Net Ordnerstruktur und setzen der benötigten Umgebungsvariablen
- Datenvorbereitung (Entpacken und vorbereiten der dataset.json-Datei)
- Datensatzkonvertierung mithilfe von nnU-Net
- Vorbereitung des Trainings via nnU-Net
- Training mit nnU-Net
- Inference mit nnU-Net
- Visualisierung der Ergebnisse
- Auswertung der Ergebnisse

Da für die Verwendung von nnU-Net eine gewisse Ordnerstruktur benötigt wird, wird diese zu Beginn erstellt. Zusätzlich werden für die Pfade dieser, die von nnU-Net benötigten, Umgebungsvariablen gesetzt. Der bereits hinterlegte Datensatz wird entpackt und die benötigte dataset.json-Datei wird, wie in Abschnitt 2.2 beschrieben, vorbereitet.

Die dann folgenden Arbeitsschritte wurden mithilfe von nnU-Net durchgeführt. Der Datensatz wird durch den Aufruf

```
1 nnUNet_convert_decathlon_task -i PATH_TO_RAWDATA_FOLDER
```

so konvertiert, dass dieser für nnU-Net nutzbar ist. Der Parameter „PATH_TO_RAWDATA_FOLDER“ gibt dabei den Pfad zum Verzeichnis der Rohdaten an.

Durch

```
nnUNet_plan_and_preprocess -t TASK_NAME_OR_ID
```

wird der Datensatz analysiert und die regelbasierten Parameter ermittelt. „PATH_TO_RAWDATA_FOLDER“ spezifiziert dabei die Tasknummer welche über den Verzeichnisnamen festgelegt wird.

Durch

```
nnUNet_train \
  CONFIGURATION \
  TRAINER_CLASS_NAME \
  TASK_NAME_OR_ID \
  FOLD\
  /
```

wird das Training gestartet. In dieser Arbeit erfolgte der Aufruf

```
nnUNet_train 3d_fullres nnUNetTrainerV2 1 0
```

wodurch das Training mit der „3D-Full-Resolution“ Konfiguration und der Trainerklasse „nnUNetTrainerV2“ für die Task-ID „1“ gestartet wurde.

Für die gewählte Trainerklasse erfolgte nun ein Training für 1000 Epochen wobei dieses nach 239 Epochen, aus zeitlichen Gründen, händisch unterbrochen wurde. Alternativ hätte eine eigene Trainerklasse erstellt werden können, die eine geringere Anzahl von Epochen trainiert hätte. Auf diesen Hinweis bin ich jedoch erst später gestoßen und da das Unterbrechen keine Auswirkung auf das resultierende Modell hat wurde dies nicht weiterverfolgt. Aufgrund der Unterbrechung mussten jedoch die gespeicherten Modelle umbenannt werden (siehe Jupyter Notebook).

Im nächsten Schritt erfolgt bereits die Verwendung des trainierten Modells. Durch

```
nnUNet_predict \
  -i INPUT_PATH \
  -o OUTPUT_PATH \
  -t TASK_NAME_OR_ID \
  -tr TRAINER_CLASS_NAME \
  /
```

können nun die gewünschten Label für die Eingabebilder vorhergesagt werden.

Mit dem Befehl

```
nnUNet_evaluate_folder \
  -ref LABELS_PATH \
  -pred PREDICTIONS_PATH \
```

```
4   -l LABEL_INDEX1 LABEL_INDEX2 ... \
5   /
```

können die ermittelten Labels durch einen Vergleich mit den den originalen Labels evaluiert werden.

## 3.2 Monai SegResNet

### 3.2.1 Allgemein

SegResNet von Myronenko, 2018 ist ein Segmentierungsansatz der auf einer Encoder-Decoder basierten CNN-Architektur aufsetzt.

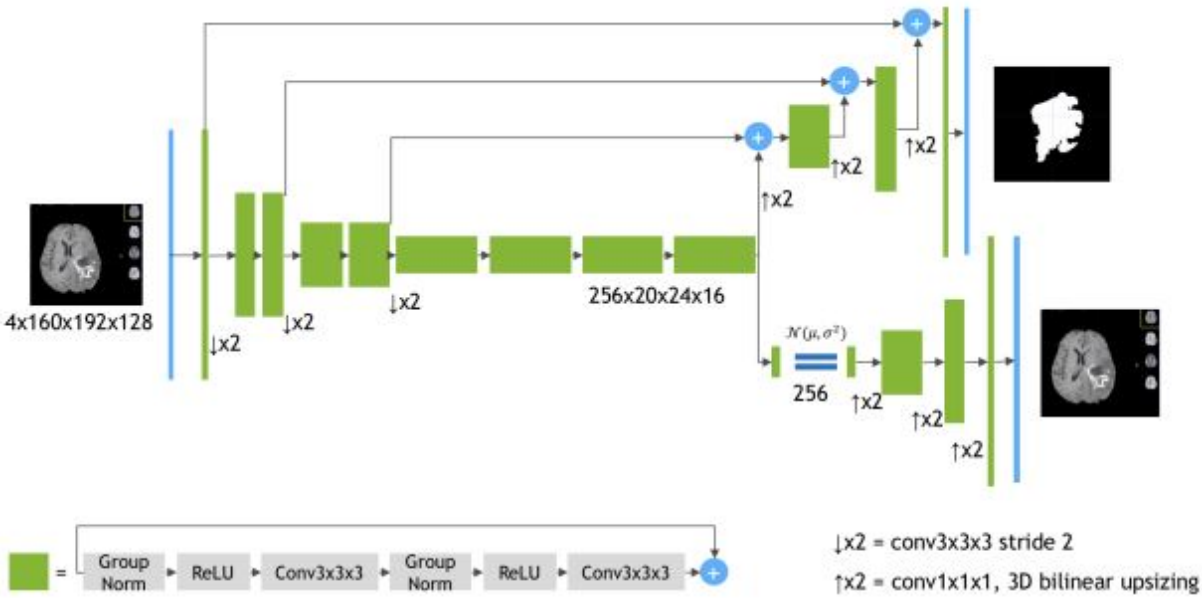


**Abbildung 3.2:** Schematische Visualiserung der Netzwerkarchitektur von SegResNet Quelle: Myronenko, 2018, S. 3

In Abbildung 3.2 sieht man eine schematische Visualisierung der Architektur. Die Eingabe ist ein Vier-Kanal 3D MRT Bild gefolgt von einer 3x3x3 3D Convolution mit 32 Filtern. Die grünen Blöcke stellen ResNet-ähnliche Blöcke mit GroupNorm normalisierung dar. Im Originalpaper von Myronenko, 2018 wurde das Bild zunehmend um zwei Dimensionen verkleinert und die Anzahl der Features um zwei erhöht. Der Decoder-Teil ist ähnlich dem Encoder, jedoch mit einem Block pro Dimension. Jede Decoderstufe beginnt mit einer Vergrößerung, welche die Anzahl der Features halbiert (durch 1x1x1 Convolutions) und mithilfe von 3D bilinear upsampling die Anzahl der Dimensionen verdoppelt, gefolgt von einer Addition der Encoder-Outputs derselben räumlichen Ebene. Die Ausgabe des Segmentierungsdecoder hat dieselbe Größe wie das ursprüngliche Bild und dieselbe Anzahl an Features wie die Eingabe. Durch eine weitere 1x1x1 Convolution wird das Bild in drei Kanäle transformiert gefolgt von einer

Sigmoid-Funktion. Der VAE (variational auto-encoder)-Branch rekonstruiert das Eingabebild und wird nur während des Trainings zur Regularisierung des Encoders benutzt. Grund für die Nutzung des Auto-Encoder-Branches ist die limitierte Verfügbarkeit der Daten.

### 3.2.2 Anwendung

Im Folgenden gehe ich auf die Anwendung des SegResNet in der Studienarbeit ein. Dafür wurde ein Jupyter Notebook erstellt, das alle Arbeitsschritte zusammenfasst. Jedoch verwende ich nicht die ursprüngliche Implementierung von Myronenko, 2018, sondern die Implementierung des Monai-Frameworks. Dabei wurde zur Hilfe das „Brain tumor 3D segmentation with MONAI“-Tutorial von Monai verwendet.

#### Dataloader

Die benötigten Daten wurden mithilfe der Monai CacheDataset-Klasse und der DataLoader-Klasse geladen. Die CacheDataset-Klasse besitzt einen Cache-Mechanismus der Daten lädt und deterministische Datentransformationen darauf cached. Daraus resultiert ein deutlich schnelleres Laden der Daten beim Training.

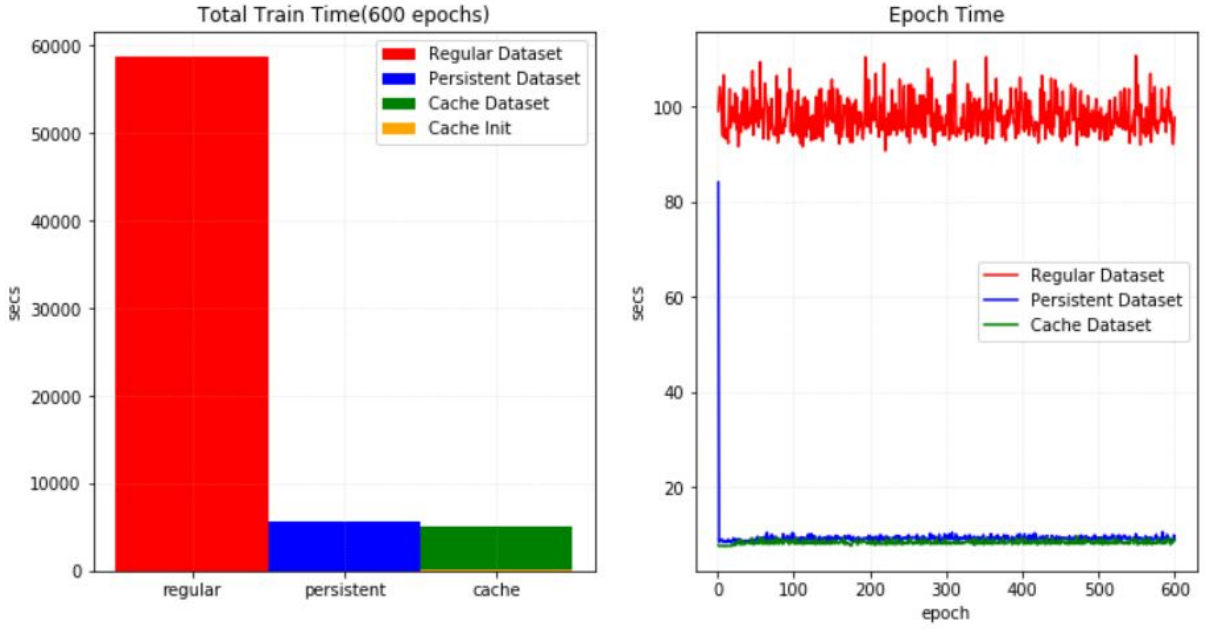


**Abbildung 3.3:** Visualisierung der Trainingsgeschwindigkeit mit verschiedenen Dataloadern Quelle: „PersistentDataset, CacheDataset, and simple Dataset Tutorial and Speed Test“, 2020

#### Transforms

Bei der Datenaugmentierung wird zuerst sichergestellt, dass beim Bild der Channel an erster Stelle steht. Dies gewährleistet die Funktion EnsureChannelFirstd. Danach werden die Labels in 3 Kanäle aufgeteilt. Durch die Orientationd-Funktion werden

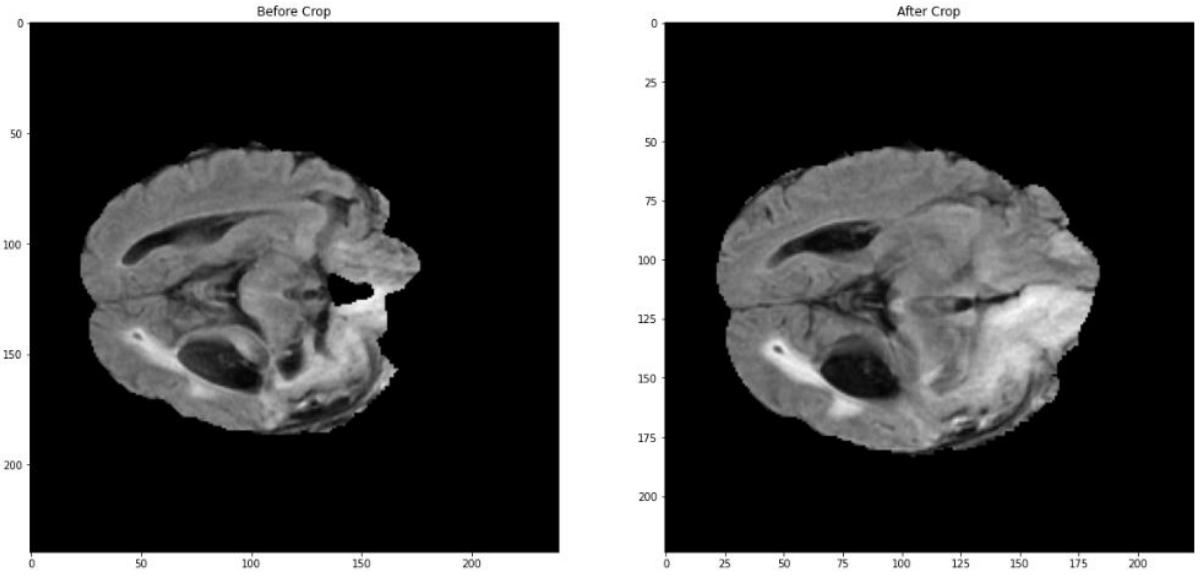


**Abbildung 3.4:** Bild vor und nach dem RandSpatialCrop

Bild und Label in ein normalisiertes Koordinatensystem transformiert. Daraufhin werden beide, durch Spacingsd, in dieselbe Pixeldimension gebracht. Danach wird ein 224x244x144 Bereich des Bildes mit zufälligem Mittelpunkt ausgeschnitten. Da die wichtigen Bereiche des Bildes im Zentrum liegen und das Bild vor dem zuschneiden eine Form von 240x240x155 hat, gehen keine wichtigen Elemente des Bildes verloren und dennoch unterscheidet sich das Bild jedes Mal vom Original. Abbildung 3.4 zeigt das Ergebnis des Zuschneidens. Zu beachten ist, dass das Zuschneiden auch auf der Z-Achse geschieht und dadurch diese nicht in beiden Bildern gleich ist.

Nach dem RandSpatialCropd erfolgt ein zufälliger Flip, durch RandFlipd, auf jeder Achse. Der Zufall besteht hierbei darin, dass der Flip zu 50% durchgeführt wird.

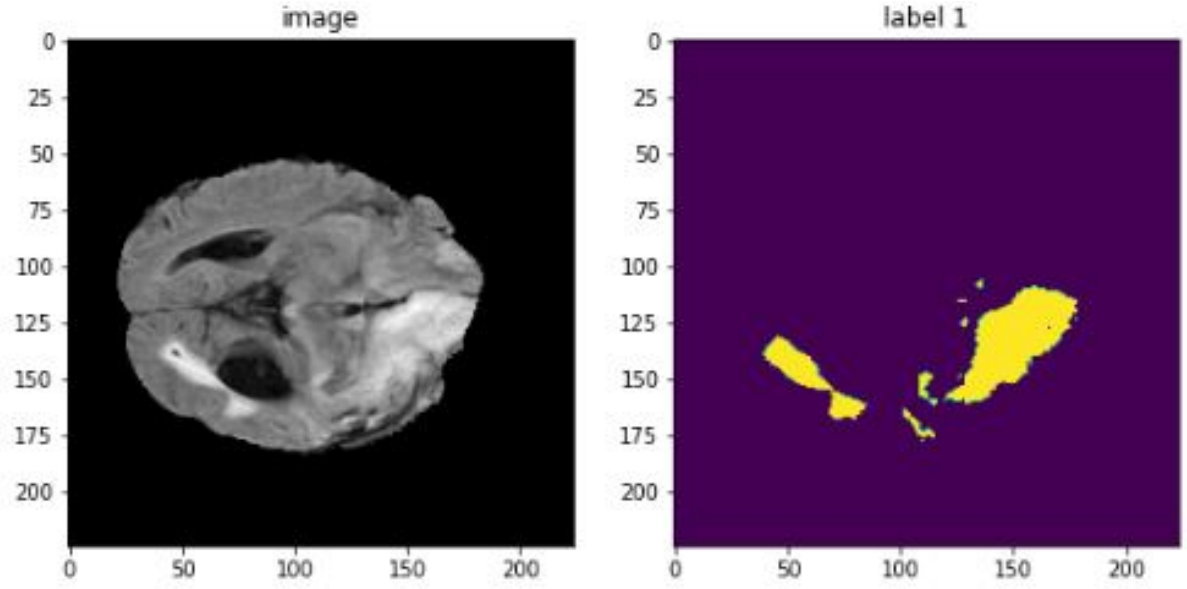


**Abbildung 3.5:** Bild und Label 1 vor dem Flip

In den Abbildungen 3.6, 3.7, 3.8 sieht man die Ergebnisse jedes Flips, wenn dieser durchgeführt wird.

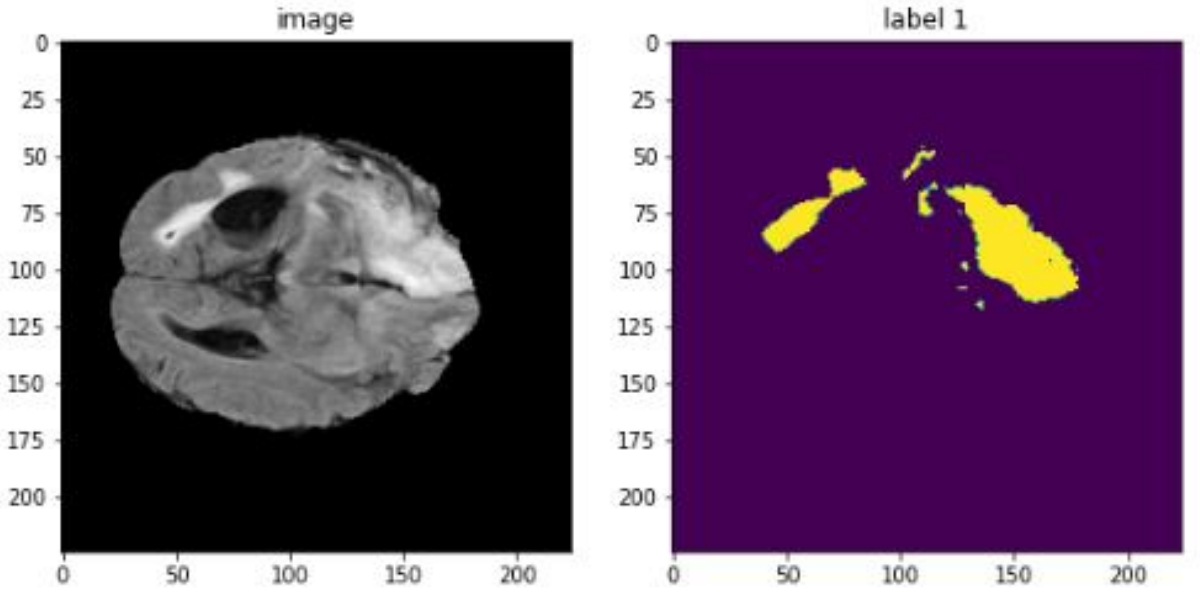


**Abbildung 3.6:** Bild und Label 1 nach dem Flip auf Achse 0

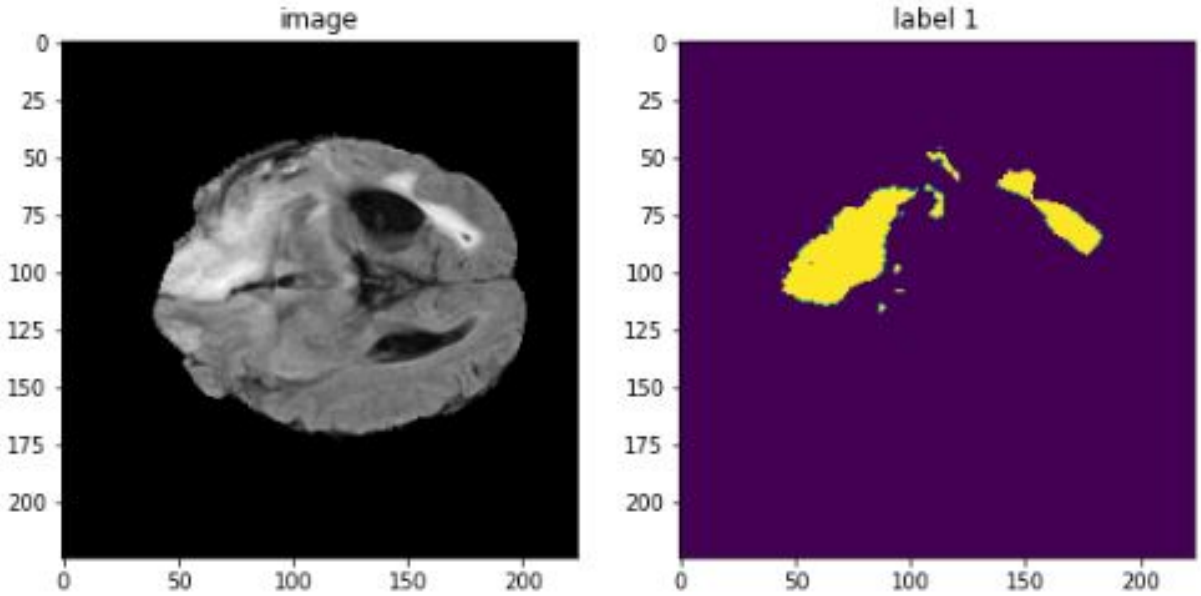


**Abbildung 3.7:** Bild und Label 1 nach dem Flip auf Achse 1

In den letzten drei Schritten wird die Intensität des Bildes transformiert. Zuerst wird die Intensität durch NormalizeIntensity normalisiert. Danach wird die Intensität zufällig skaliert und im letzten Schritt um einen zufälligen Offset verschoben. In Abbildung 3.9 sieht man die Auswirkung der Normalisierung gut. Die zufälligen Transformationen sind mit dem bloßen Auge jedoch kaum zu erkennen.

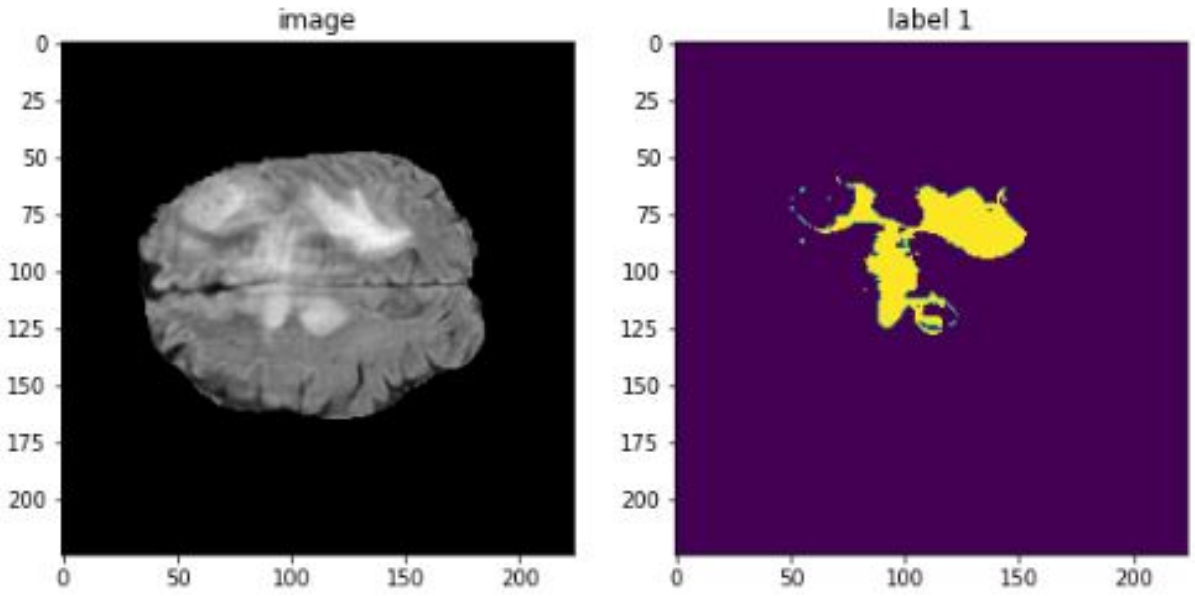


**Abbildung 3.8:** Bild und Label 1 nach dem Flip auf Achse 2

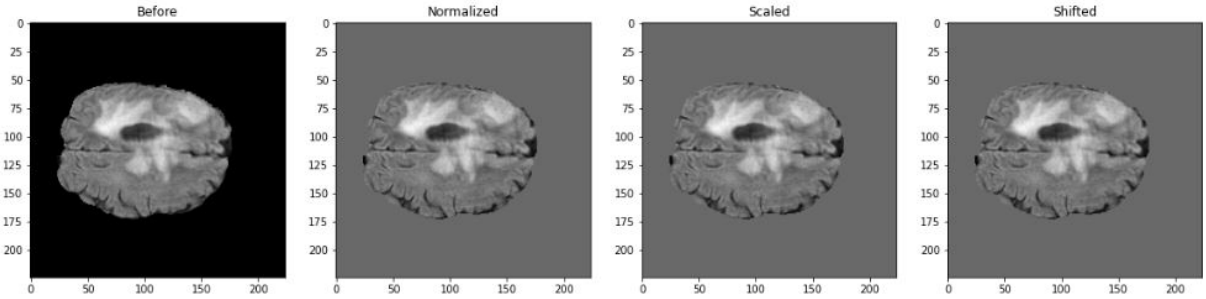


**Abbildung 3.9:** Intensitätstransformationen

### Training

Zum Training wurde ein SegResNet mit den Parametern

- blocks_down=[1,2,2,4]
- blocks_up=[1,1,1]
- init_filters=16
- in_channels=4
- out_channels=3
- dropout_prob=0.2

initialisiert. Die Blocks UP und Down entsprechen dabei dem Schema aus Abbildung 3.2. Die initialen Filter geben dabei die Anzahl der Ausgabekanäle des initialen Convolutionlayers an. Die Eingabekanäle entsprechen hier den 4 Eingabebildern, also den 4 Modalitäten des 3D-Bildes. Die Ausgabekanäle entsprechen hierbei den drei Labels.

Dropout_prob gibt hierbei an, mit welcher Wahrscheinlichkeit ein Element genulled wird. Dies ist eine Methode zur Regularisierung und soll Overfitting vermeiden. Als Loss-Funktion wird der Dice-Loss verwendet sowie Adam als Optimizer. Danach wird das Training mit einer Batchsize von zwei gestartet. Im Training wird für jede Epoche der durchschnittliche Loss sowie der Dice-Score jedes Labels des Validierungsdatensatzes gespeichert. Sofern das Modell einen besseren Dice-Score erreicht, wird das Modell zwischengespeichert. Wie bei nnU-Net wurde wieder mit 239 Epochen trainiert um im nächsten Kapitel die beiden miteinander vergleichen zu können.

# Kapitel 4

# Ergebnisse und Diskussion

Nachdem nun auf demselben Datensatz ein Modell mit nnU-Net und eines mit SegResnet trainiert wurden sollen im Folgenden die Ergebnisse aufgezeigt werden.

| Modell | Dice-Score je Label | | | |
|---|---|---|---|---|
| • | Edema | Non-Enhancing Tumor | Enhancing Tumor | Duchschnitt |
| nnU-Net | 0.7770 | 0.5954 | 0.8003 | 0.7243 |
| SegResNet | 0.7851 | 0.5645 | 0.7901 | 0.7133 |

**Tabelle 4.1:** Dice-Scores der Modelle je Label

Wie in Tabelle 4.1 zu sehen ist, erzielt nnU-Net ein leicht besseres Ergebnis. Lediglich bei der Segmentierung des Edema liegt das SegResNet minimal vorne. Jedoch denke ich das mit weiteren Optimierungen ein besseres Ergebnis erzielt werden könnte. Dies war mir im Rahmen der Studienarbeit aus zeitlichen Gründen leider nicht mehr möglich. Bevor ich das SegResNet trainiert hatte, habe ich einige Versuche mit dem SwinUNetr-Modell unternommen. Durch den begrenzten GPU-Speicher der Hochschulrechner war es mir mit diesem Modell leider nicht möglich vernünftige Ergebnisse zu erzielen weshalb ich diesen Ansatz verwarf. Jedoch stieß ich auch mit dem SegResNet an meine Grenzen, bis ich Zugriff zum GPU-Server erhielt. Leider ging dabei viel Zeit verloren die ich gerne noch in Anpassungen des SegResNet gesteckt hätte. Hinzu kommt, dass das Training des SegResNet etwa 24h+ dauerte und ein durch einen Neustart einige Ergebnisse verloren gingen und ich daher das Training erneut gestartet habe.

In Abbildung 4.1 werden in der oberen Reihe die ursprünglichen Label angezeigt, in der unteren Reihe die Ergebnisse des Modells. Dasselbe gilt für Abbildung 4.2.

In Abbildung 4.3 kann man den Trainingsverlauf des SegResNet sehen. Deutlich wird das die Ergebnisse erst ab Epoche 50 annehmbar sind. Ab Epoche 200 ist keine Verbesserung mehr zu erwarten. Das beste Ergebnis des SegResNet wurde in Epoche 197 erreicht. Zu nnU-Net liegen mir diese Daten leider nicht vor.

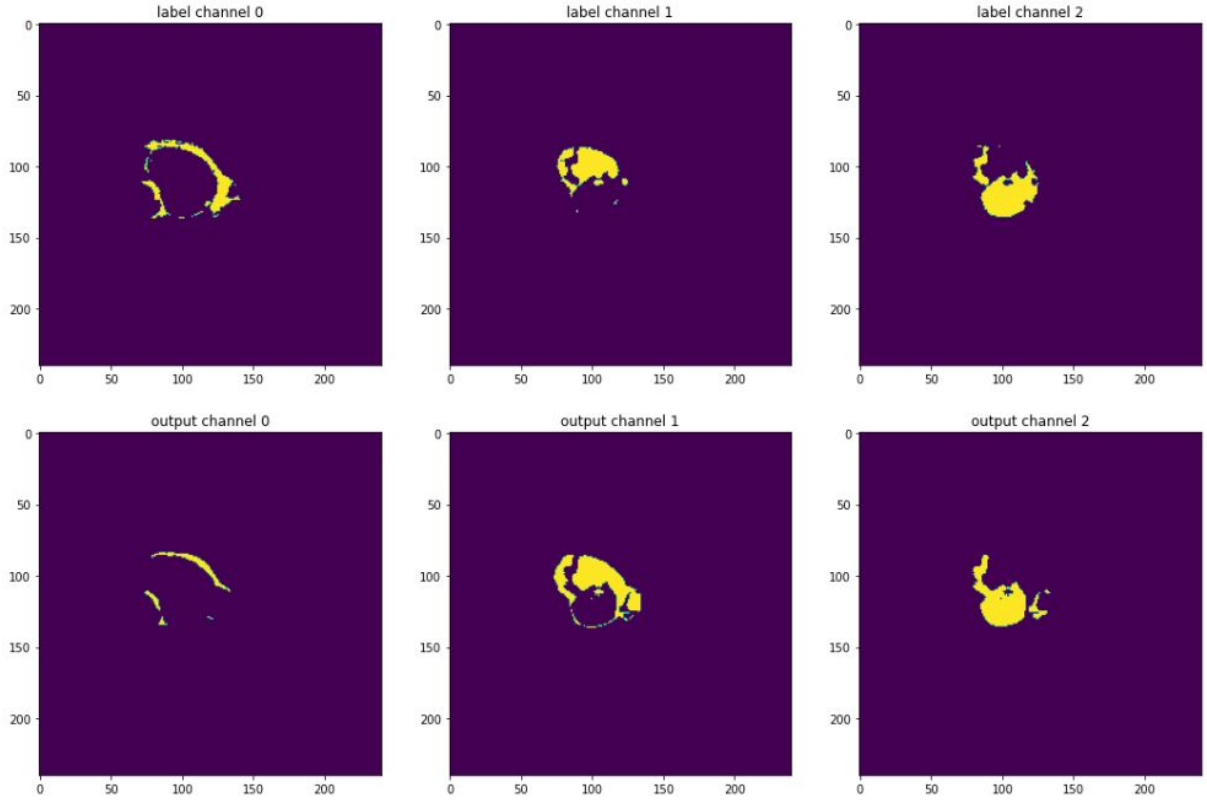


**Abbildung 4.1:** Ergebnis nnU-Net

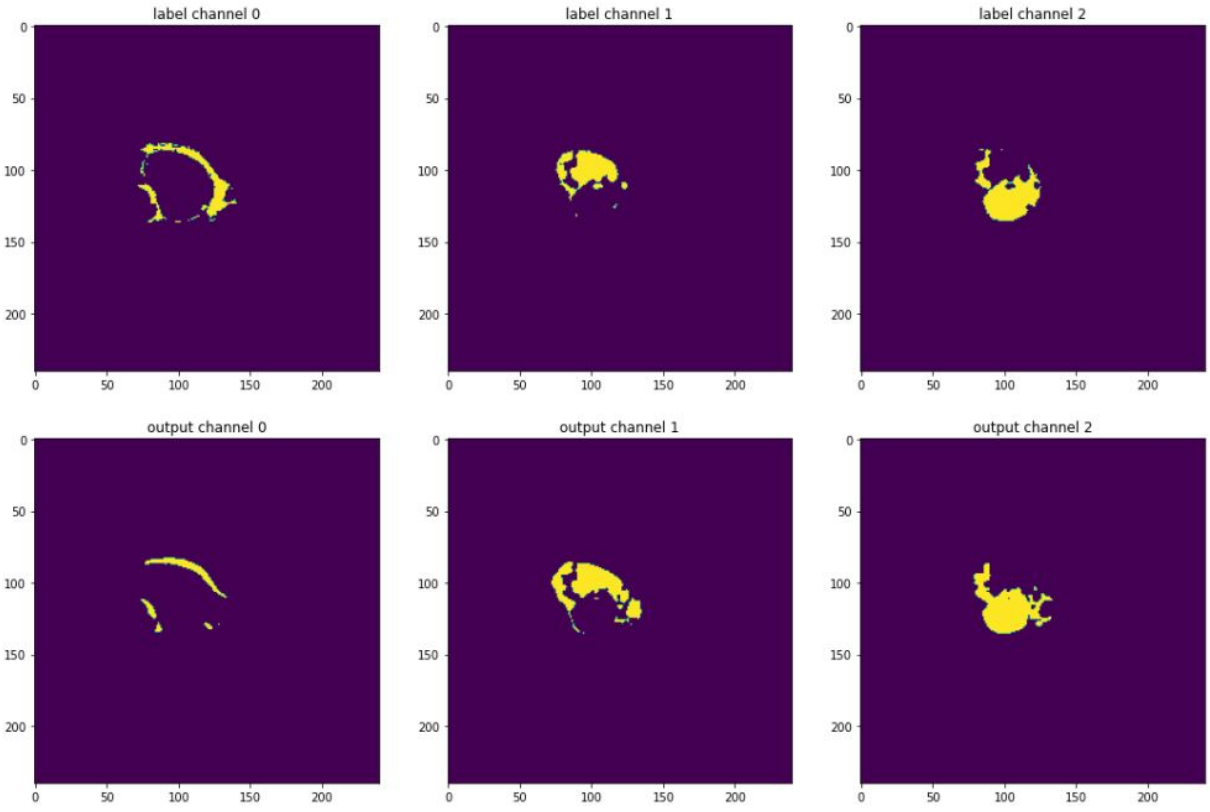


**Abbildung 4.2:** Ergebnis SegResNet

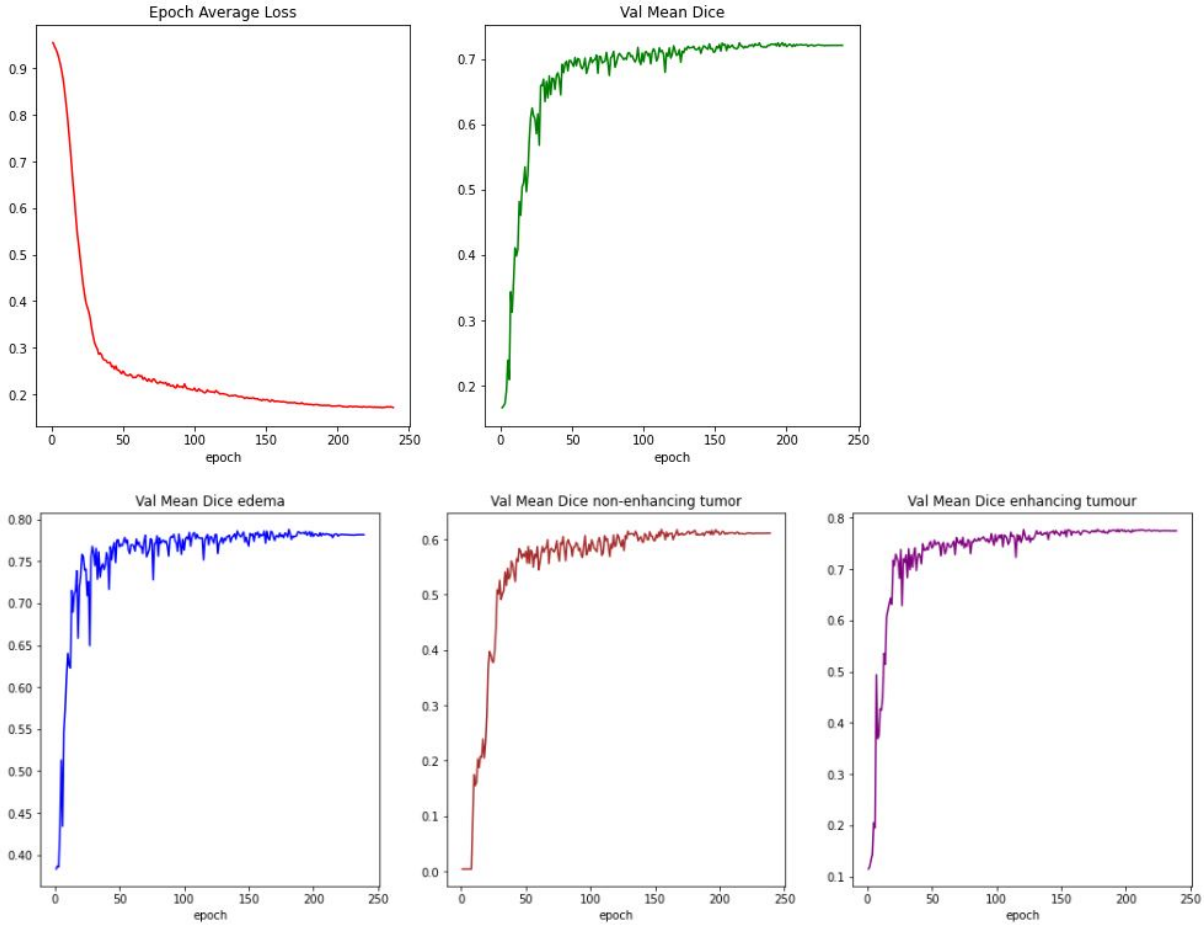


**Abbildung 4.3:** Trainingsverlauf SegResNet

# Literatur

Antonelli, M., Reinke, A., Bakas, S., Farahani, K., AnnetteKopp-Schneider, Landman, B. A., Litjens, G., Menze, B., Ronneberger, O., Summers, R. M., van Ginneken, B., Bilello, M., Bilic, P., Christ, P. F., Do, R. K. G., Gollub, M. J., Heckers, S. H., Huisman, H., Jarnagin, W. R., … Cardoso, M. J. (2021). The Medical Segmentation Decathlon. https://doi.org/10.48550/ARXIV.2106.05735

Brain tumor 3D segmentation with MONAI. (2020). *Project Monai*. Zugriff 29. Juni 2022 unter https://github.com/Project-MONAI/tutorials/blob/main/3d_segmentation/brats_segmentation_3d.ipynb

Isensee, F., Jaeger, P. F., Kohl, S. A. A., Petersen, J. & Maier-Hein, K. H. (2020). nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation [Journal Article Research Support, Non-U.S. Gov't]. *Nature methods*, *18*(2), 203–211. https://doi.org/10.1038/s41592-020-01008-z

Isensee, F., Petersen, J., Klein, A., Zimmerer, D., Jaeger, P. F., Kohl, S., Wasserthal, J., Koehler, G., Norajitra, T., Wirkert, S. & Maier-Hein, K. H. (2018). nnU-Net: Self-adapting Framework for U-Net-Based Medical Image Segmentation. http://arxiv.org/pdf/1809.10486v1

Lüth, C., sten2lu & Bungert, T. (2022). nnU-Net Workshop. Zugriff 29. Juni 2022 unter https://github.com/IML-DKFZ/nnunet-workshop

Myronenko, A. (2018). 3D MRI brain tumor segmentation using autoencoder regularization. http://arxiv.org/pdf/1810.11654v3

PersistentDataset, CacheDataset, and simple Dataset Tutorial and Speed Test. (2020). *Project Monai*. Zugriff 29. Juni 2022 unter https://github.com/Project-MONAI/tutorials/blob/main/acceleration/dataset_type_performance.ipynb