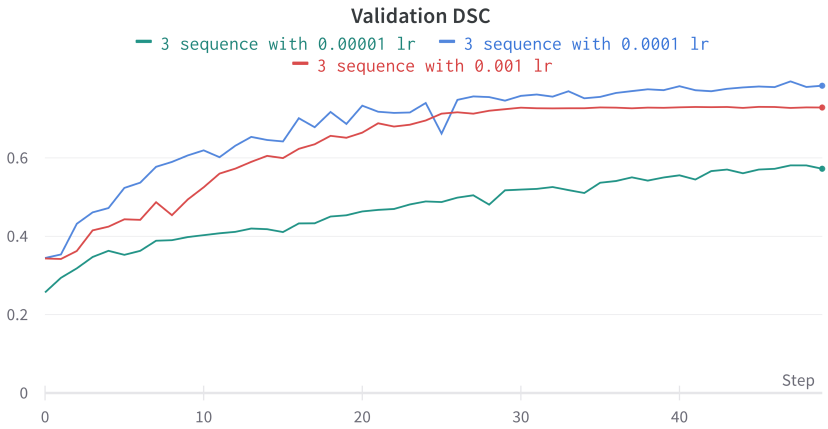
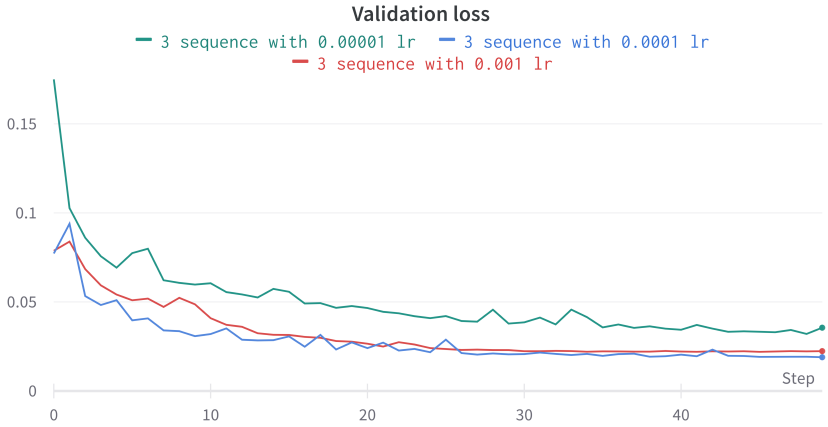
Appendix A

Training graphs

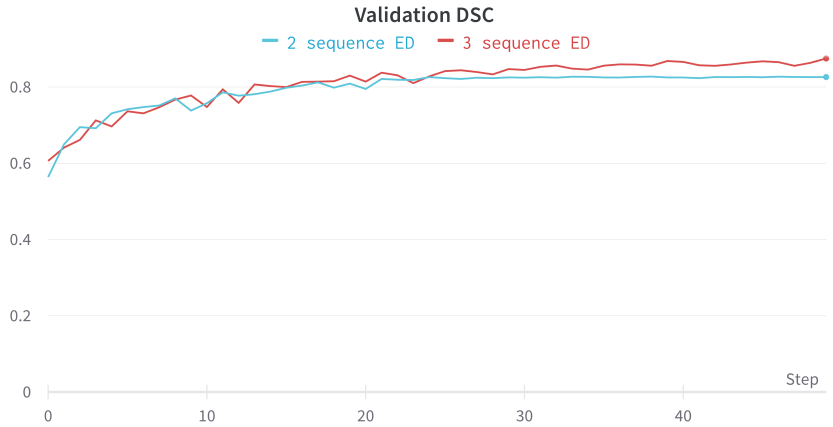


(a) Evolution of validation DSC

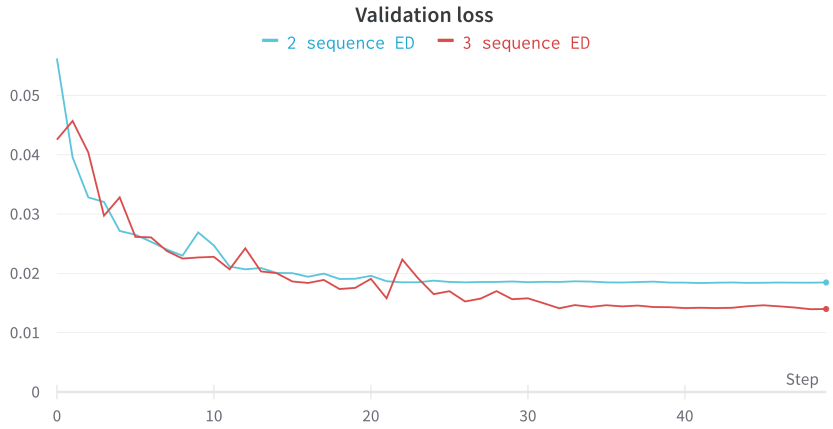


(b) Evolution of validation loss

Figure A.1: Comparison between validation DSC and loss for the models trained with three channels and three different learning rates, where the model with 0.001 LR achieved a validation DSC of 0.7286, the model with 0.0001 LR achieved 0.7842 and the model with 0.00001 LR achieved 0.5723.



(a) Evolution of validation DSC



(b) Evolution of validation loss

Figure A.2: Comparison between two sequence and three sequence training for the ED subregion, where the model trained for segmenting the ED subregion achieved a final validation DSC of 0.8748 compared to the 0.8260 achieved by the two sequence model.

Appendix B

Technical documentation

B.1 Requirements

For the implementation of our work we used the version 3.8 of the Python programming language. As for the most important libraries we used:

- tensorflow - version 2.7.0
- keras - version 2.7.0
- numpy - version 1.21.0
- nibabel - version 3.2.2
- matplotlib - version 3.5.0

All of the required libraries with their respective versions are present in the digital medium in file `requirements.txt`.

B.2 Source code

The source code is divided into multiple files, and their layout, as well as a short description for each one, can be found in the Appendix C.

One of the more important files is the `train.py`, which incorporates the training of the models. The main part of the file is a loop which defines training parameters for each subregion. The main parts of the loop can be found in the listing below:

```
if config['num_classes'] == 4: subregions = [0]
else: subregions = [1, 2, 3]

for subregion in subregions:
    # setting the training and valitation generators
    train_img_datagen = BratsGen(...)
    val_img_datagen = BratsGen(...)

    # defining and compiling the model
    model = unet(
        img_height=128, img_width=128, img_depth=128,
        img_channels=config["img_channels"],
        num_classes=config["num_classes"]
    )
    model.compile(optimizer=optim, loss="categorical_crossentropy",
        metrics=metrics)

    # training the model
    history = model.fit(
        train_img_datagen,
        steps_per_epoch=steps_per_epoch,
        epochs=config["epochs"],
        verbose=1,
        callbacks=[get_callbacks(...), WandbCallback()],
        validation_data=val_img_datagen,
        validation_steps=val_steps_per_epoch
    )
```

Listing B.1: The main loop in train.py

The idea behind the loop was to make the training of multiple models for individual subregions seamless, so when we are segmenting all subregions at once, the loop will only do one iteration. For the training of the models, we used a method `model.fit()` provided by TensorFlow.

Another important part are the scripts used for evaluation in the `evaluation/` directory. Both `evaluation.py` and `evaluation_separate.py` are very similar. The main part is the `model_eval(my_model, testing_dataset)` method that runs the evaluation of the specified model, where we make use of the `model.predict(test_image` method provided by TensorFlow, which generates a prediction, or in our case, a segmentation mask, for the given input image. Again we loop over the testing dataset and call this method for each image and calculate the metrics for the given segmentation. In the end, we print out the mean values for the metrics and save the individual slices as `.png` files for both the best five images and the worst five. The only difference in `evaluation_separate.py` is that before calling `model.predict(test_image)`, we need to merge the individual segmentations created by the models. To merge the segmentations, we use a method called `combine_models(model1, model2, model3, img)`, which generates individual segmentations and combines them into a newly merged segmentation, which is subsequently returned by the method.

Another files contains helper methods used by the mentioned files and are not supposed to be executed. One type of these files are utility files in the `utils/` directory, which includes methods for data processing and data manipulation (`data_processing.py`) and custom callbacks (`callbacks.py`). Then our implementation also uses a custom data generator defined as a class in `generator.py`. The function of the custom data generator is to provide batches of preprocessed and optionally augmented data to the training algorithm. Another helper file

is a `model.py`, which defines a TensorFlow model with U-Net architecture, later used in training. Furthermore, the last files used in our implementation are the `losses.py`, which defines our custom loss functions and `metrics.py`, which defines custom metrics.

B.3 Usage

The training and evaluation scripts were built for the Azure ML services, so the workflow outsite the Azure services might feel a little sluggish and was not properly tested. For the execution of the scripts it is required to have the mentioned requirements. The best way is to set up a virtual environment with:

```
python3 -m venv venv
```

and activating it with:

```
source venv/bin/activate
```

After activating the enviroment, you can install the requirements with:

```
pip install -r requirements.txt
```

The individual scripts can be run by the `python` command as follows:

```
python train.py
```

You can also provide a path to your dataset with the `--data_path` argument. The dataset has to include a `train/` and `val/` directories for training and `test/` for validation. The scripts will use the provided sample files if the data path is not specified. By default, the scripts use the predefined configuration that would need to be changed in the source code. For the training script, the configuration is in the `config` dictionary. The evaluation scripts would run the best respective models provided in the `models/` directory. In order to use different models, you would need to change the configuration variables on the top of the evaluation files.

Appendix C

Contents of the digital medium

Registration number of the thesis in the information system: FIIT-101018-97014

Contents of the digital medium (ZIP archive):

```
src/ - source code for the implementation
  evaluation/ - scripts used for evaluating models
    evaluation.py - for models segmenting whole tumour
    evaluation_separate.py - for models segmenting individual sub.
  utils/ - auxiliary methods used during training or evaluation
    callbacks.py - defines custom callbacks
    data_processing.py - methods used for data processing
    utils.py - miscellaneous methods
  generator.py - custom data generator
  losses.py - defines custom loss functions
  metrics.py - defines custom metrics
  model.py - defines U-Net model used in training
  train.py - script used for training
```

```
requirements.txt - files containing a list of required libraries
models/ - best models from each experiment
  separate/ - contains models segmenting individual subregions
    3ch/
      model_NCR.h5 - model segmenting NCR subregion
      model_ED.h5 - model segmenting ED subregion
      model_ET.h5 - model segmenting ET subregion
  model_3ch_aug_e4.h5 - model segmenting all subregions
BraTS2021/ - contains sample MRI scans that we used in our work
  train/ - sample training scans
    BraTS2021_00002/
    BraTS2021_00003/
  val/ - sample validation scans
    BraTS2021_00019/
    BraTS2021_00006/
  test/ - sample testing scans
    BraTS2021_01621/
    BraTS2021_01627/
docs/ - pdf versions of the final work
  main_part.pdf - main part of the work without the appendices
  appendices.pdf - only the appendices
  final.pdf - both the main part and appendices
```

We did not include the whole dataset used during our experiments because it has 13.4 GB. The original dataset can be downloaded from kaggle[1].

Name of the submitted archive: BP_JakubPovinec.zip.

[1] https://www.kaggle.com/datasets/dschettler8845/brats-2021-task1

Appendix D

Project task schedule

D.1 Winter semester

1^{st}-2^{nd} week	Specification of the domain and the dataset
3^{th}-4^{th} week	Studying basics of neural networks
5^{th}-6^{th} week	Exploring the dataset and experimenting with NNs
7^{th}-8^{th} week	Studying the state-of-the-art methods and related work
9^{th}-10^{th} week	Experimenting with U-Net and the dataset
11^{th}-12^{th} week	Working on the analytical part of the document

During the winter semester, our focus was primarily on studying convolutional neural networks and gaining experiences with neural networks in general. We managed to finish most of the analytical part of the work and successfully created our first model.

D.2 Summer semester

1st-2nd week	Defining the architecture and training first models
3th-4th week	Working on the first experiment
5th-6th week	Incorporating the Azure ML services into our workflow
7th-8th week	Working on the second experiment
9th-10th week	Evaluating the conducted experiments
11th-12th week	Finishing the document

In the summer semester, we deepened our knowledge of convolutional neural networks. We improved the model from winter and designed multiple methods for the segmentation of gliomas. Subsequently, we were able to finish and evaluate said methods, after which we focused primarily on finishing the document.