Joseph Rivera

Dr. Chen Chen

Medical Image Computing

2/28/2022

Assignment 2 Report

Introduction

In this assignment, I trained U-Net for brain tumor segmentation. MATLAB was used to prepare the model. The necessary dependencies are MATLAB 2021b and the deep learning toolbox, computer vision toolbox, and image processing toolbox. Please see my public github page here for the code. The primary script is the .mlx jupyter-notebook-like file that shows all of my figures in-line. (JosephRivera1/braintumorsegmentation (github.com) JosephRivera1/braintumorsegmentation (github.com)).

Visualization of BRATS with ITK-SNaP

ITK-SNAP is a tool used for accurately segmenting 3-D volumes. Today we will examine the Brain Tumor Segmentation (BRATS) dataset, which contains full 3-D volumes of MRI scans of 750 brains. **Figure 1** shows the direct output of segmenting BRATS_1 which has all 3 labels: label 1 (red) is edema, label 2 (green) is a non-enhancing tumor, and label 3 (blue) is an enhancing tumor. Another example is shown in **Figure 2**.

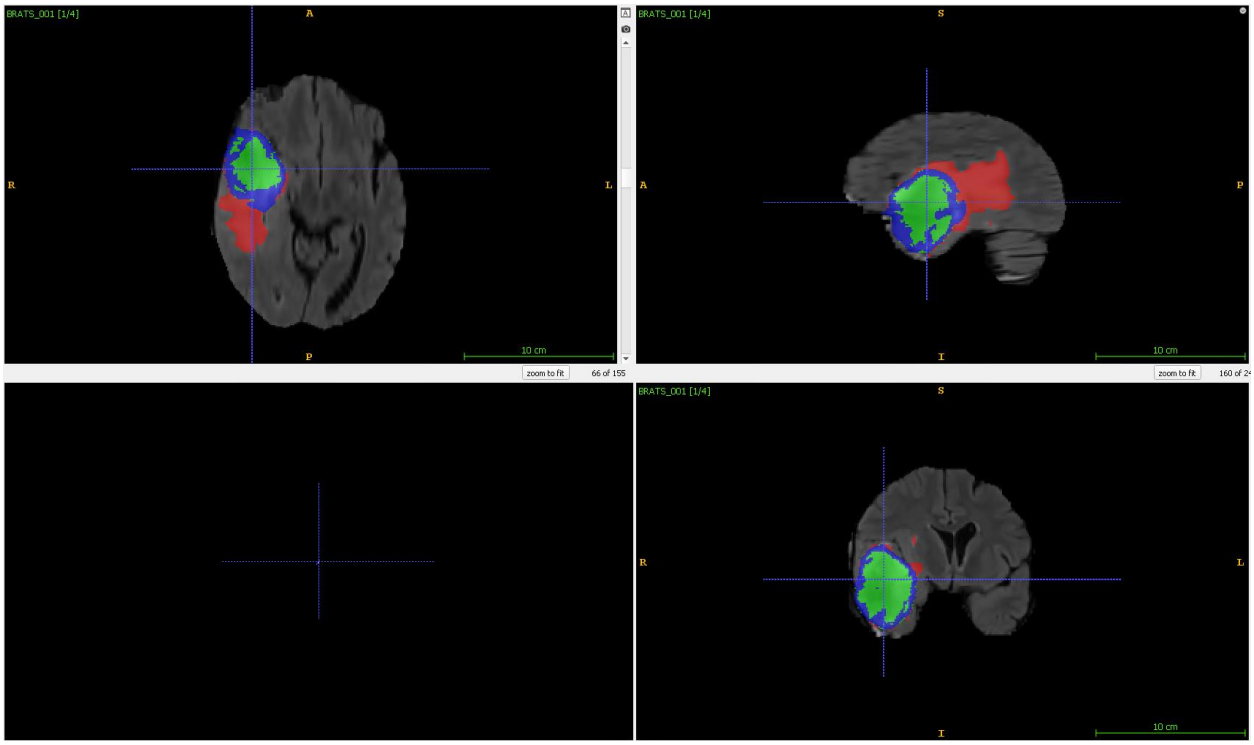

Figure 1: BRATS_1 BRAIN MRI with Segmentation Mask

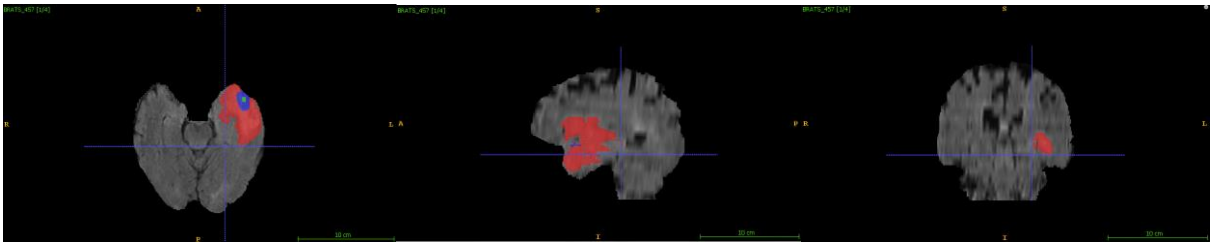

Figure 2: Overlay of Segmentation Mask on BRATS_457

Matlab Set Up

This implementation is adapted from a MATLAB example using just two classes for 3D segmentation. MATLAB contains useful tools to visualize our data, like labelvolshow() as seen in **Figure 3** below. It also natively reads the files with niftiread().

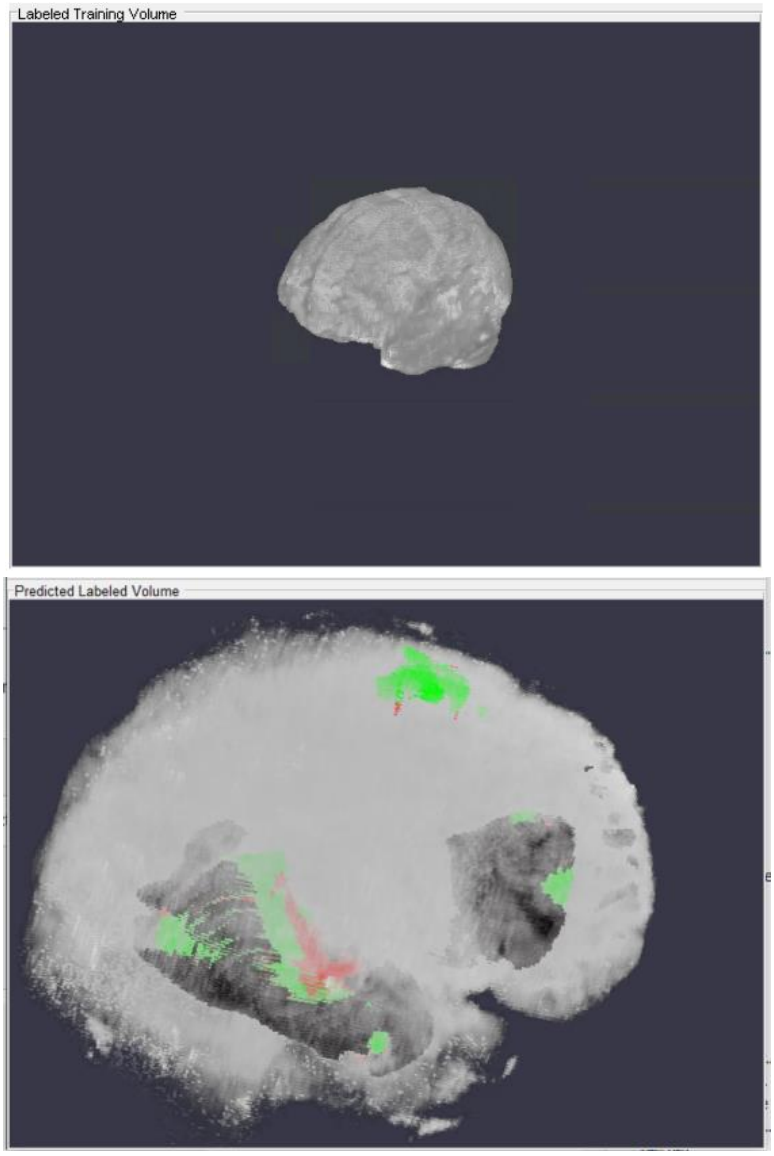

Figure 3: Sample 3D volume from the Matlab Image Datastore with the BRATS data

5-fold Cross-Validation

I made 5 datasets using a 20% increment as the testing data. Dataset1 uses 0-20%, Dataset2 uses 20%-40%, and so on to Dataset 5. **Figure 4** shows the best result from experiment 4 of training (blue) and validation (black dots) accuracy scores. The final validation accuracy was 95.91%. Training had to be cut short of a targeted 50 epochs due to time constraints and only having access to one NVIDIA RTX 5000.

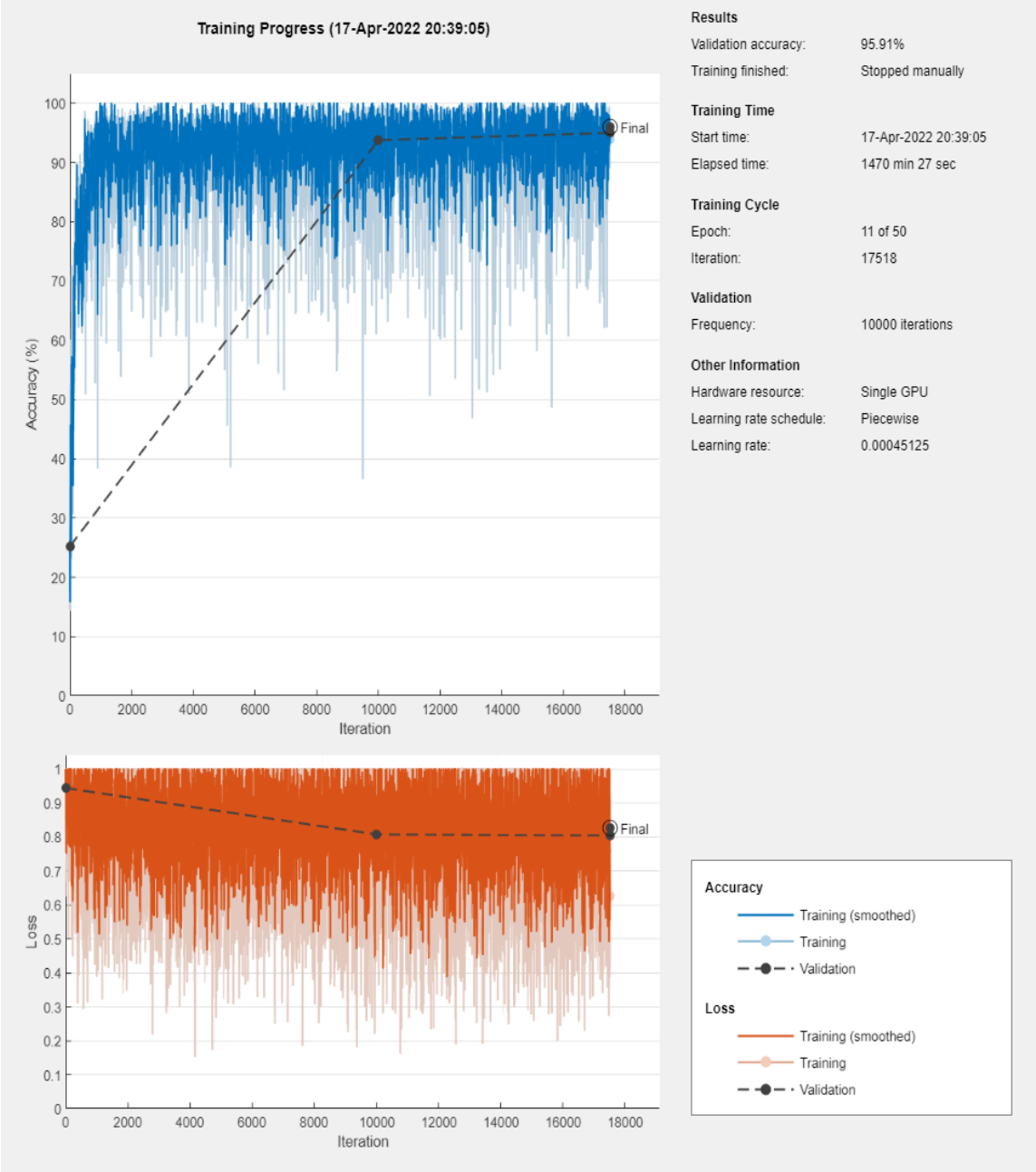

Figure 4 Training Plot

U-Net 3D Architecture

We will use the U-Net 3D architecture to segment the entire volume and perform a 5-fold cross-validation on the training set (imagesTr). I'm sticking with the default MATLAB network size of 64x64x64, and we'll use a patch extraction technique that resizes the 132x132x132 patch to the network input with augmentations e.g. flips and rotations.

Training Details

To train on these large volumes, we employ a random patch extraction technique that samples parts of the volume for training. I stuck with default training parameters like Adam optimizer to 50 epochs maximum with a learning rate of 5e-4 and a scheduled drop in learning rate every 5 epochs by 0.95.

Results

Below shows the results of our full 3-D UNet, in **Figure 5** you can see the qualitative results of the algorithm segmenting a brain. You can see the ground truth on the left and the prediction on the right. Although the algorithm highlights common areas it does misclassify the left edema which would be a dangerous false alarm. There is also incorrectly predicted edema speckling throughout the brain but those areas are very small.

The dice score takes into account high class imbalances and is simply twice the intersection over the total number of pixels. **Figure 6** shows the scoring results between background and other classes. All in all the dice scores returned 99.78 for background, 61.93 for edema, 0.43 for a non-enhancing Tumor, and 75.46% for an enhancing tumor. Although classes were confused, the results highlight the characteristic regions of the targeted brain tumors. I would let the algorithm train the full amount next time as the training plots show a convergence, but quite noisy training.

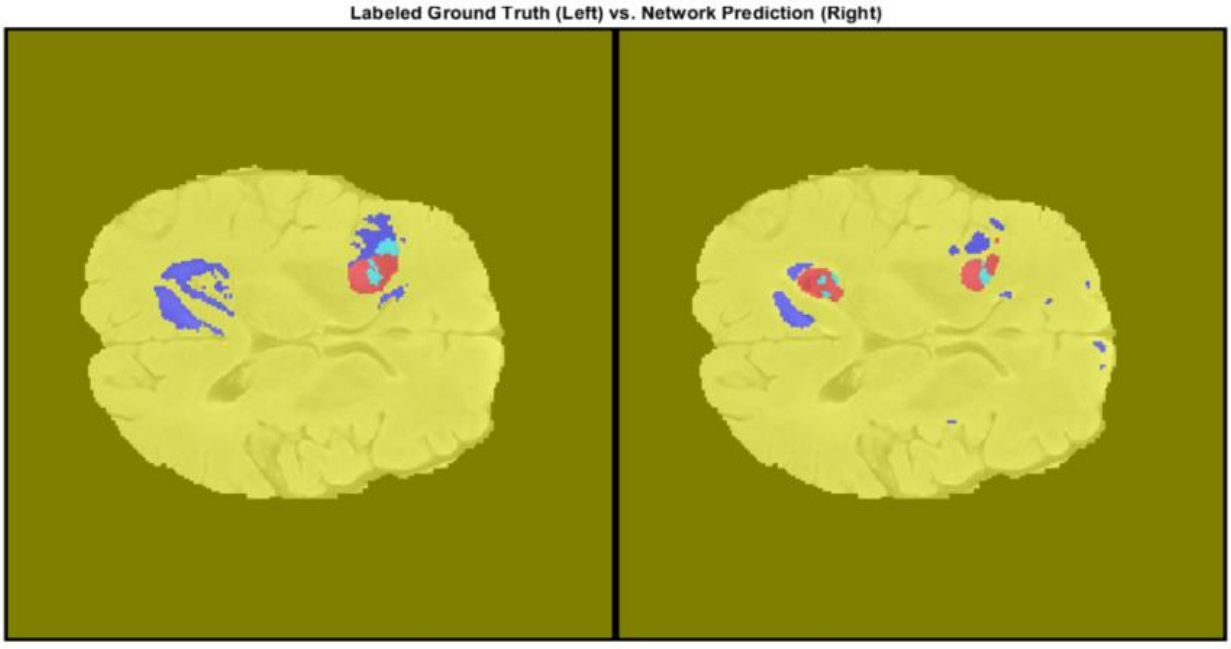

Figure 5: Qualitative Sample of Brain Tumor Segmentation

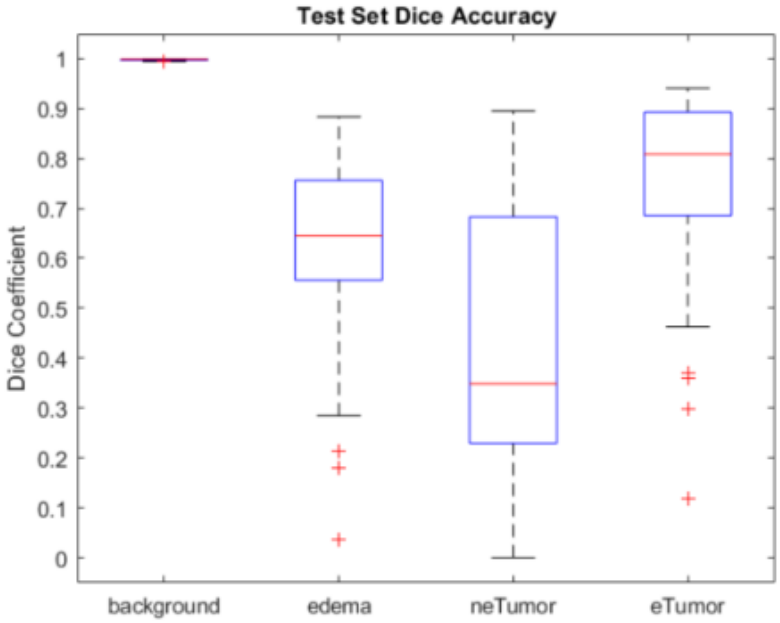

Figure 6: Box-Plot of Dice Scores

Conclusion

I was able to do the full segmentation using a 3-D UNet and score using the dice method. I had trouble adapting the model from 2 classes to 4 and that caused the algorithm to fail to converge due to poor pre-processing with the patch extraction technique. If I had access to better GPUs locally I would have been able to train the algorithm much faster as well, I was only able to finish 11 epochs in 24.5 hours, each full experiment would have taken at least 5 days to train if I had not cut the training short. There's certainly much to be desired from the performance but the preliminary results show excellent background segmentation and hint at a good performance given adequate training time. The data augmentations may have also contributed to the noisy training plot which hampers convergence time and may not have been ideal for quick experiments.