Brain Tumor Segmentation Using Diferent U-Net on MRI

Names: Chan Kaiyi, Duan Lingbo, Jovan Yap, Shu Bohan, Xu Boyu
Uniquenames: kachan, popcorna, jovanyap, shubohan, xuboyu
University of Michigan, Ann Arbor

Abstract

Accurate segmentation of brain tumors from MRI scans is pivotal for advancing diagnostic and therapeutic approaches in neuro-oncology. This study leverages the 3D U-Net architecture to enhance the segmentation process, aiming to provide precise detection and quantification of brain tumors. Utilizing the BRATS dataset, the research begins with extensive preprocessing, followed by rigorous accuracy assessments of the U-Net model. To counter data imbalance, augmentation strategies are implemented, with further refinements via advanced filters and feature extraction techniques. The study explores the potential integration of cutting-edge models such as UnetFormer and MedSegDiff, compare the performance of 2D U-Net model and 3D U-Net model and evaluates the efficacy of model ensembling for improved accuracy. Initial findings reveal superior performance of models trained on a diverse array of images from Brats2021, highlighting the significance of comprehensive training data. This work promises to be a substantive contribution to the field of medical imaging and artificial intelligence, with implications for enhanced patient care in neurological disorders.

1. Introduction

The accurate segmentation of brain tumors from Magnetic Resonance Imaging (MRI) scans is a critical challenge that underpins the effectiveness of neurological diagnostics and treatment strategies. The present study, anchored in this imperative, applies the 3D U-Net [7], a deep learning architecture renowned for its proficiency in medical image segmentation. This research endeavors to improve the delineation of tumorous tissues within MRI scans, aiming to bolster the detection and monitoring of brain tumors.

In pursuing this objective, the researchers have secured access to the Brain Tumor Segmentation (BRATS) datasets [3, 4] and the corresponding U-Net model with pre-trained weights [6, 7]. Initial efforts encompass a comprehensive preprocessing of the MRI data to condition it for optimal model performance. The U-Net model's accuracy will be meticulously evaluated post-application on the test images, utilizing both dice score and accuracy metrics.

Acknowledging the prevalent issue of data imbalance in medical imaging, the study incorporates data augmentation techniques to address this challenge, thereby enhancing the model's training efficacy. Further refinements include the evaluation and integration of advanced filtering and feature extraction methodologies during the preprossessing phase.

Preliminary results have indicated a notable improvement in model performance with increased data diversity, as demonstrated by the enhanced outcomes of models trained on the Brats2021 dataset compared to those conditioned on Brats2020 [1, 2]. This observation underscores the significant impact of extensive and varied training data on model proficiency in brain tumor segmentation.

Research Questions: 1. To segment the tissues captured from brain MRI images to detect and quantify the existence and possibly growth of brain tumors. 2. To compare the performance of different U-Net Models and investigate the factors that influence the segmentation quality and efficiency of brain tumor

2. Background

Magnetic Resonance Imaging (MRI) has been a pivotal tool in diagnosing and monitoring neurological disorders, particularly brain tumors. Its non-invasive nature and high spatial resolution allow for detailed visualization of soft tissues, which is crucial for accurate tumor detection and assessment.

Recent studies have integrated advancements such as deep convolutional networks, and more specifically, adaptations of the U-Net model, to improve segmentation tasks. Comparative analyses, like the Brain Tumor Segmentation (BraTS) challenge, have been instrumental in pushing the boundaries of what these models can achieve, providing a benchmark for current methods and highlighting areas for improvement.

3. Methodology

To answer our research questions, we will examine four different models together with their training strategies. We implement three models: (i) A model basing on simplest principle of 2D U-Net without any attachment (ii) An improved version of 2D U-Net through our exploration. (iii) A model basing on the principle of 3D U-Net. We compare their results with each other and also with a previous developed model named SwinUNETR [6].

3.1. Dataset

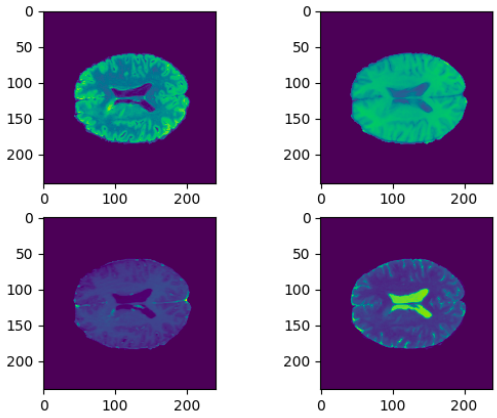

Figure 1. DataSet Example

The BraTS 2020 dataset [3] consists of a training set and a validation set. The training set is organized into several sub-folders, each containing image slices representing four different aspects of brain tumors: T1-weighted (t1), T1-weighted post-contrast (t1ce), T2-weighted (t2), and FLAIR (fluid-attenuated inversion recovery). Each type of imaging data includes 155 transverse scan layers of the brain, with each layer being a 240x240 pixel image. Figure 1 shows an example of one layer (4 slices) of a brain. Additionally, each sub-folder contains a corresponding ground truth segmentation ('seg') of the brain tumor, also comprising 155 layers.

However, the validation dataset lacks the ground truth segmentation data. The absence of this segmentation data precludes its use for validating model performance in traditional supervised learning frameworks. As a result, an alternative approach involves partitioning the training data into two subsets. These subsets are then utilized respectively as the new training and validation datasets to enable effective model evaluation and tuning.

For the BraTS 2021 dataset, there are 1251 folders with the 4 different aspects of the brain tumor and the ground truth ('seg').

3.2. Self-Designed Models (U-Net)

3.2.1 Image prepossessing

An image in BraTS 2020/2021 [3] [4] contains 2D-slices of the brain, each with shape $240 * 240$. We use the transformer interface from ***torch.transformers*** to upscale each image to shape $256 * 256$ and crop at the center of the image to shape $224*224$. Next, basing on the Dataset interface from ***torch.utils.data.Dataset***, we implement a class named BraTDataset, refering to [5]. For 2D U-Net in this project, we employ 2D Convolution Layers. For each transverse layer of the Brain Tumor, we stack 4 corresponding slices in the order ('t1', 't1ce', 't2', 'flair'). Our target is to use these first four aspects to predict a segmentation of the image to determine the position of tumors, so the 4-slice stack constitutes the input to our model and the corresponding 'seg' slice is the ground truth. The BraTDataset class constructs two element tuples containing the stacked images (4-slice stack) and labels ('seg' slice). Afterwards, we set batch size 4 (more details later) and apply the DataLoader interface from ***torch.utils.data*** to get the train loader and validation loader, with shuffle set to be true for both of the loaders. Here each batch has the shape $(4, 4, 224, 224)$ and each label has shape $(4, 1, 224, 224)$.

For our 3D U-Net model, the 3D image are up-scaled to shape $155 * 256 * 256$ and crop to shape $144 * 224 * 224$. Instead of only stacking one slice, we stack all 155 slices in each type of 't1', 't1ce', 't2' and 'flair'. After setting batch size 1, the batch has the shape $(1, 4, 144, 224, 224)$ and each label has shape $(1, 1, 144, 224, 224)$

3.2.2 Basic 2D U-Net

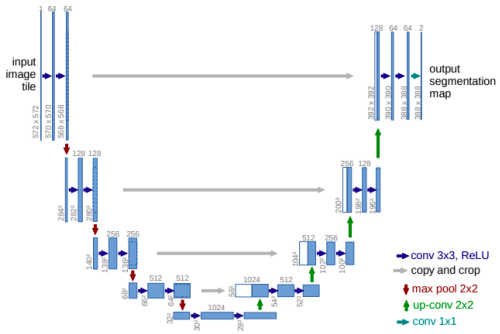

Figure 2. U-Net [7]

The U-Net architecture [7] is a convolutional network specifically designed for the task of biomedical image segmentation. It is structured into two main components: the Decoder (down-sampling) and the Encoder (up-sampling).

In the Encoder segment, each down-sampling step comprises two convolutional layers with a kernel size of 3x3,

stride of 1, and padding of 1 to maintain the same spatial dimensions ('same padding'). These convolutional layers are interleaved with two ReLU activation layers. This sequence is followed by a max pooling layer with a kernel size of 2x2 and a stride of 2, which reduces the spatial dimensions by half. As the image progresses through successive down-sampling stages, the feature depth increases exponentially through the series: 64, 128, 256, 512, and finally 1024. This exponential increase in feature depth allows for more complex features to be captured at each level.

The Decoder part consists of sequential up-sampling steps. Each up-sampling step includes one up-convolution layer (implemented as either ***nn.ConvTranspose2d*** or ***nn.Upsample***) followed by two convolutional layers with a kernel size of 3x3, stride of 1, and padding of 1, alternated with two ReLU layers. The up-sampling process also integrates features from the corresponding down-sampling stages via skip connections. This is achieved by concatenating the current layer's output with the output from the corresponding down-sampling layer (***torch.cat([x, skipConnection], dim=1)***), enabling residual learning and helping the network learn more robust features and maintain details by utilizing both high-level and low-level information.

3.2.3 Self-improved 2D U-Net

In our implementation of the Original U-Net for biomedical image segmentation, we encountered issues with gradient vanishing when utilizing a relatively small dataset. This is detailed further in the **Results** section. To combat the gradient vanishing issue - characterized by an extremely slow decrease in IoU (Intersection Over Union) loss, we introduced ***nn.BatchNorm2D*** layers between each convolution layer pair. *Batch normalization layer* helps in maintaining robust gradient flow throughout the network, thus alleviating the problem of gradient vanishing.

Additionally, we switched our loss function from IoU loss to Dice Loss and the Dice Loss we defined are shown in Equation (1). 10^{-30} here is to avoid the divided-by-zero error. Dice loss is more effective for decreasing the gap between all positive instances predicted by the model and the ground truth. It is particularly suitable for semantic segmentation tasks as it directly optimizes the Dice Similarity Coefficient (DSC), providing a robust mechanism for training.

$$\text{Dice Loss} = \frac{2. * intersection + 10^{-30}}{label.sum() + pred.sum() + 10^{-30}} \quad (1)$$

We also improve the training strategy by implementing transfer learning in our training regimen. Initially, the U-Net model is trained on 32 data sub-folders for 30 epochs, and the model weights are saved into *.pt* files [1]. Training then continued on another set of 32 sub-folders with the model initialized using the previously saved weights from the best model. This approach helps in generating a more effective initialization and adapting the model to new data by utilizing previously learned features.

Furthermore, we modify the Dice Loss function by adding more penalties to false positive predictions. Specifically, we employ a relatively larger loss when the intersection of predicted and ground truth areas is small, and a relatively smaller loss when the intersection is large. This adjustment helps in fine-tuning the model's accuracy and its sensitivity to segmentation errors. Equation (2) shows a modified Dice Loss with penalty.

$$\text{Dice Loss} = \frac{2. * penalty * intersection + 10^{-30}}{label.sum() + penalty * pred.sum() + 10^{-30}} \quad (2)$$

Moreover, in our preprocessing steps, we decide eliminate the utilization of image normalization, which was part of our earlier image transformation practices. We posit that those normalization might disrupt the gradient flow, which is crucial for the learning process. By eliminating this transformation, we aim to maintain the structural integrity and orientation of the images, thereby supporting more consistent gradient behavior during model training.

3.2.4 Self-designed 3D U-Net

The basic structure of this 3D U-Net is very similar to the structure of above mentioned self-improved 2D U-Net, except that ***nn.Conv2D*** is changed to ***nn.Conv3D***. Since the input of 3D model is memory-consuming, we set the feature depth to increase through the series: 8, 16, 32, 64 in the encoding process, which is much smaller than those in 2D U-Net. Also, we use both Dice Loss with and without penalty to train this model.

3.2.5 Hyperparameter setting

For 2D U-Net model, our training process contains four different parts. In the first part, we train our U-Net model from scratch on data sub folders from index 0000 to 0031 and evaluate the model on data sub folders from 0320 to 0351. The parameters settings are displayed in the following table:
In this part, we initialize all weights uniformly within $[0, 1]$ and set all biases to 0 and we set 0 penalty in the Dice Loss. For the second part, we train our U-Net model with no penalty in the Dice loss on data sub folders from 0032 to 0063 and evaluate the model on data sub folders from 0336 to 0367. The parameter sets are the same as Table 1, except that we only train for 13 epochs. The model is initialized by

[1] You can find those .pt files in Appendix: Google Drive/Self-Improved U-Net Training Checkpoint/

Table 1. parameters in the first part of training

Params	Value
batch size	4
learning rate	0.1
weight decay	0
num of epoch	30
optimizer	*torch.optim.SGD*
scheduler	*torch.optim.lr_scheduler.CosineAnnealingLR* $T_{max} = 100$

the parameters of the best model (highest Dice score) in the previous part.[2] The third part is topology to the second part, except that we use penalty 2 and only train for 8 epochs. For the last part, we train our U-Net model on data sub folders from 0032 to 0063 again and evaluate the model on data sub folders from 0336 to 0367. We assign:

Table 2. parameters in the third part of training

Params	Value
batch size	4
learning rate	0.05
weight decay	0
num of epoch	20
optimizer	*torch.optim.SGD*
scheduler	*torch.optim.lr_scheduler.StepLR* step_size = 3, gamma = 0.8

After the training process, we choose the model with best performance on validation data as the final model. In addition, we try three different penalty (3, 5, 8) in Dice Loss and compare their influence on validation performance, which will be further discussed in Result section. For the 3D-Unet model, we only conduct Training part I.

3.2.6 Training

During our training process, we employ a custom-designed data loader that is configured with batch size 4. Throughout each epoch, we systematically process all batches. For each batch, predictions (*pred*) are obtained from our model, and these predictions are subjected to our modified Dice Loss for the backward pass.

Furthermore, given the high prevalence of empty data and ground-truth pairs within our dataset, we have implemented a strategy to selectively skip some of these pairs. Specifically, we identify these empty pairs and choose to drop them with a 50% probability. This method helps in balancing the training data and reducing the bias towards non-informative data points.

In terms of learning rate schedules, we employ both cosine and linear schedules to optimize our training dynamics. During the initial phase of our training, we use a cosine schedule. This schedule assists in escaping local minima by adjusting the learning rate according to a cosine curve, which can lead to more robust exploration of the parameter space. In the ultimate phase, we switch to a linear schedule. This approach provides a steady decrease in the learning rate, which is aimed at refining the learning process and enhancing the accuracy of the model.

[2]In the following discussion, we omit this sentence since we always choose the weights of best model to process.

3.3. SwinUNETR Model from Monai

Upon finding the SwinUNETR model and discovering that it has a promising average dice score of 0.7828 on the BraTS 2021 dataset, we decided to train this model on the BraTS 2020 dataset to see if we could potentially perform model ensembling with our self-designed model that is trained on the BraTS 2020 dataset for improved accuracy. Furthermore, we performed training on BraTS 2021 dataset as well to analyze the different effects on the dataset on the performance of the UNET model.

3.3.1 Data preprocessing

Since there are major differences between the BraTS 2020 and 2021 datasets such as the dataset size (BraTS 2021 dataset has 1251 training files while BraTS 2020 only has 369) and availability of segmented data (BraTS 2020 validation data has no segmentation labels), we made adjustments to the model to work for BraTS 2020 training data. For example, the model uses a 5-Fold Cross-Validation, but because we have limited training files, we modified the json file that the model uses to divide the training files into different folds so that it uses 3-Fold Cross-Validation instead. In addition, due to limited memory usage on Google Colab, we decreased the feature size from 48 to 36 to not run out of memory, which can reduce computation time but also decrease the accuracy of the model. We also tried to further reduce memory usage by decreasing the batch size from 2 to 1.

To further explore the potential of this model, we decided to use cross-test the models and the SwinUNETR model trained on the BraTS 2021 dataset was tested on the BraTS 2020 dataset, and the SwinUNETR model trained on the BraTS 2020 dataset was tested on the BraTS 2021 dataset [4].

3.3.2 SWIN U-Net Model architecture

This is the reference architecture for the SWIN U-net model [6] that was used. The U-net model has an encoder and decoder model for down sampling and up sampling to produce the image segmentation.

Table 3. SWIN U-Net architecture

Layer	Dimensions
Input	128,128,128,4
End of Encoder	4,4,4,768
Start of upsampling	8,8,8,384
output	128,128,128,3

3.3.3 Hyperparameter setting

A smaller batch size has been chosen to meet the resource limitation on google colab. Specific parameter setting are displayed in Table 4 and 5.

Table 4. parameters for model trained on BraTS 2020 dataset

Params	Value
batch size	1
learning rate	1e-4
weight decay	1e-5
num of epoch	24
optimizer	*torch.optim.AdamW*
scheduler	*torch.optim.lr_scheduler.CosineAnnealingLR* $T_{max} = 100$

Table 5. parameters for model trained on BraTS 2021 dataset

Params	Value
batch size	1
learning rate	1e-4
weight decay	1e-5
num of epoch	3
optimizer	*torch.optim.AdamW*
scheduler	*torch.optim.lr_scheduler.CosineAnnealingLR* $T_{max} = 100$

3.3.4 Training

During the training process, the dataloader written in SWIN UNet training pipeline by MONAI [6] is used to load the BraTS image data into the model configured with an appropriate batch size. Throughout each epoch, the data has been preprocessed before the model makes a prediction and then the predictions are subjected to the Dice less for backwards pass. Given that some of the data is missing, checks are done to ensure that the images that do not exist will be dropped from the dataset.

For the learning rate schedules, the cosine schedule is chosen for both training processes. This schedule is useful to skip the local minima by adjusting the learning rate based on the cosine curve thus leading to a more effective training and higher probability to reach the global minima.

4. Results

Sections 4.1-4.4 are about to answer the **Research Question 1**: To segment the tissues captured from brain MRI images to detect and quantify the existence and possibly growth of brain tumors. **Section 4.5** is to answer the **Research Question 2**: To compare the performance of different U-Net Models and investigate the factors that influence the segmentation quality and efficiency of brain tumor.

4.1. Basic U-Net

For basic U-Net, we start with ***nn.MSELoss*** as our loss function and the proportion of correctly predicted points as our evaluation criteria. But since there are many zero pixels in labels, the model can earn a very high score by predicting every pixel 0. This evaluation metric cannot effectively reflect how well our model predict the area where the ground-truth is 1, so we change the evaluation metric to Dice Score (1 - Dice Loss in equation (1)). However, we find that the Dice Score for both data set (training/validation) remains extremely low and the accuracy stays like a constant without any climbing. We visualize some example prediction of U-Net model (Figure 3) and realize that the model fails to predict any brain tumor (The prediction is zero almost everywhere). After investigation, we conclude that the MSE

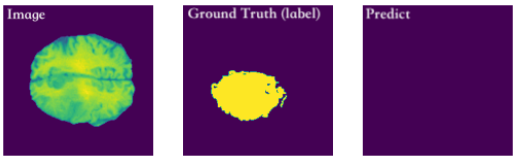

Figure 3. Result: Empty Prediction

loss function and the model architecture are improper and trigger the gradient vanishing problem. We modify the loss and the model to create a self-improved U-Net.

4.2. Self-Improved U-Net

Instead of using MSE loss, we employ the Dice Loss defined by us in equation (1). In the first part of our training, we use fixed training data (32 sub-folders from 0000 to 0031) to train our model for 30 epochs. The training/validation accuracy and dice loss are shown in Figure 4. Note that those three criteria are all in $[0, 1]$.

Basing on Figure 4, we find two interesting point. During Epoch $[12 - 14]$, the dice loss for training data reduces a lot, while the accuracy for both set drops sharply. Since we are using ***CosineScheduler*** in this part of training, this drop indicates our model is escaping from a local minima with low accuracy. During Epoch $[22 - 27]$, the dice loss for training data reduces a lot, while the accuracy for both set increases a lot. This indicates that our model finds a better spot along the loss distribution and goes into that spot.

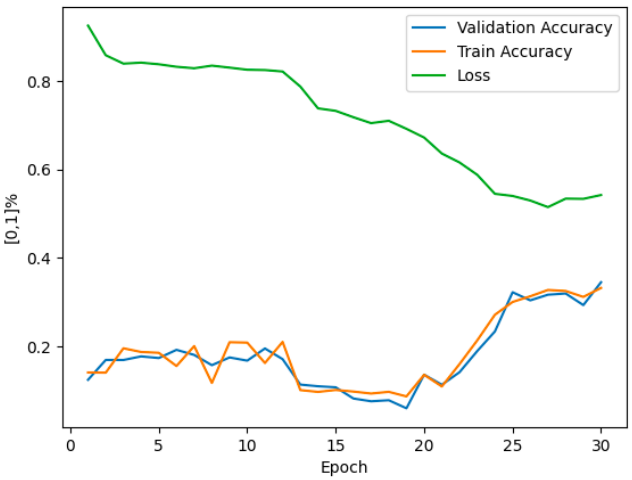

Figure 4. Training Part I - Train from Scratch

However, the model we get in part I still contain some issues. The most important problem we figure out is that in order to obtain high socre our model starts to generate false positive prediction on some images (See Figure 5). False positive prediction mean that even if the ground truth is empty, our model predicts some brain tumor. To tackle

Figure 5. Result: False Predict

those false positives, we modify our Dice Loss by adding penalty to those cases with large unions but small intersection. We modify original Dice Loss into Dice Loss $= \frac{2.*penalty*intersection+10^{-30}}{label.sum()+penalty*pred.sum()+10^{-30}}$, shown previously in Equation (2).

In part II, we use this modified Dice Loss without penalty to perform the second part of training. We change training data (another 32 sub-folders from 0032 to 0063) and transform learning for epoch $[31-43]$ (initialize weights with part I model 30) and the model 37 turns out to be the best one. Then we try to increase penalty, in training part III, we set *penalty*=2 and transform learning our previous model (initialize weights with part II model 37). We train for 8 epochs (epoch $[38-45]$) and the model in epoch 43 achieves the best performance. In part IV, we experiment different penalties. We start with weights of model 43 and the same training set, try *penalty*=$3, 5, 8$ parallelly and get the following training/validation distribution (Figure 6). In the figure, both *penalty* smaller than 5 (s.t. 3) and *penalty* larger than 5 (s.t. 8) perform worse than *penalty*=5. We

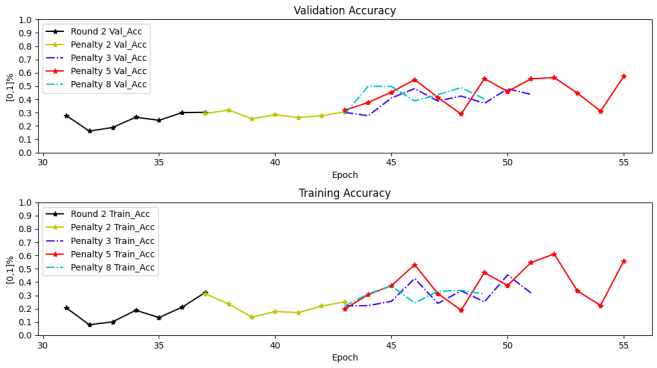

Figure 6. Training Part II, III, IV

decide to generate the final model by continuing training using penalty 5 Dice Loss. Finally, our best model (model 52) achieve:

Validation Accuracy: 0.57148
Training Accuracy: 0.48555

It is worth to mention that the Dice Score we use for evaluation is **NOT** including any penalty, since modified Dice Loss will evaluate the loss differently. Modified Dice Loss is only used for training purpose. There are some valid prediction obtained from model 52 (See Figure 7,8).

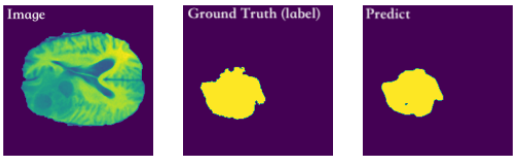

Figure 7. Good Result 1

Figure 8. Good Result 2

4.3. Self-Designed 3D U-Net

For 3D U-net, we simply train 30 epochs using Dice Loss without penalty. The training and validation performance is shown in the following figure:
The best is in epoch 30, which earns:

Validation Accuracy: 0.45129
Training Accuracy: 0.40961

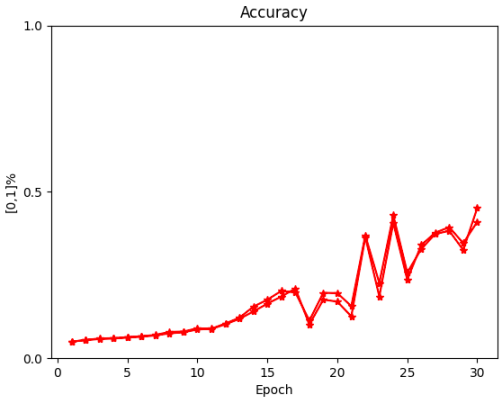

Figure 9. Dice Score of 3D U-Net model

4.4. SwinUNETR

Both models performed relatively well though the SWIN U-net model trained on 2021 BraTS dataset had a slightly higher validation accuracy. The model trained on the 2021 BraTS dataset took 1/8 the epochs of the SWIN U-net trained on the 2020 BraTS dataset to reach a relatively similar validation accuracy. Based on our analysis, it can be attributed to the difference in the size of the dataset as the 2021 BraTS dataset had more approximately 3 times more images compared to the 2020 BraTS dataset which led to a more comprehensive training.

Table 6. SWIN U-net trained on 2020 BraTS Dataset

Layer	Dimensions
Average dice loss	0.3943
Time taken for training	1600s
Total Epochs	24
Validation Accuracy	0.7596

Table 7. SWIN U-net trained on 2021 BraTS Dataset

Layer	Dimensions
Average dice loss	0.3969
Time taken for training	3928s
Total Epochs	3
Validation Accuracy	0.7740

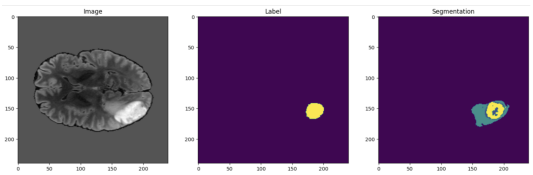

Figure 10. True positive of SWIN UNet model trained on 2020 BraTS dataset

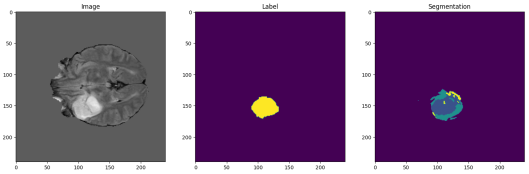

Figure 11. True positive of SWIN UNet model trained on 2021 BraTS dataset

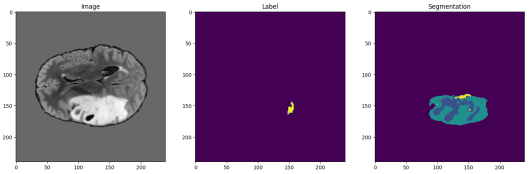

Figure 12. False positive of SWIN UNet model trained on 2020 BraTS dataset

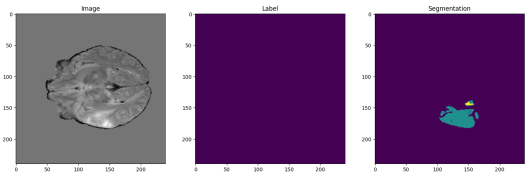

Figure 13. False Positive of SWIN UNet model trained on 2021 BraTS dataset

4.5. Comparison

4.5.1 Training Time Comparison

In the table 8, the self-improve U-Net needs $> 1h$ to train 30 epochs while SWIN U-Net only needs $< 10min$ to train 30 epochs. The reason why this happens is that self-improved U-Net uses *2D conv. layer* which splits 155 layers into 155 different inputs (e.g. 32 sub-folders of training data will generate $32 * 155$ different inputs), while SWIN U-Net uses *3D conv. layer* which combines 155 layers into 1 inputs (e.g. 1000 sub-folders of training data will only generate 1000 different inputs).

Table 8. Training Time Comparison (30 Epochs)

Model	Training Time
Self-improved 2D U-Net	$> 1h$
SWIN U-Net	$< 10min$

4.5.2 Performance Comparison

In the table 9, the validation accuracy of the self-improved 2D U-Net is lower than SWIN U-Net. **Reason 1,** Self-Improved 2D U-Net utilizes ***nn.conv2d*** layers while SWIN U-Net utilizes ***nn.conv3d*** layers, therefore SWIN U-Net will consider relationship (context) between different lay-

ers while Self-Improved U-Net do not. **Reason 2,** Self-Improved 2D U-Net only uses 32 data-subfolders to train while SWIN U-Net uses $>$ 300 data-subfolders to train, which means the training dataset for SWIN U-Net is larger than Self-Improved U-Net. Moreover, the self-designed 3D U-Net in training part I (first 30 epochs) achieves much better performance than the self-improved 2D U-Net in the first 30 epochs although it uses less parameters. This also justified that 3D U-net model can learn features from context among different layers of 3D images while 2D U-Net may not. Yet we see that the accuracy score of self-improved 2D U-Net in training part IV (52 epochs) transcends the self-designed 3D U-Net in training part I. This is probably attributed to the Dice Loss with penalty in training part III and IV. The penalty added successfully decrease the false positive the model may predict while preserving those true positive predictions. Our self-designed 3D U-Net model has lower performance than SWIN U-Net model. This is mainly because SWIN U-Net model is trained on much more images ($>$ 1000) and the self-designed 3D U-Net model is trained on only 32 images. Larger dataset can dramatically reduce the variance in the model and thus can improve the validation performance. In addition, from the validation accuracy curve in the result sections (Figure 6, 9), we see that as number epochs became larger, the curve fluctuate more. This implies that the learning rate at large epoch is too high and should decrease more as training epoch grows.

Table 9. Accuracy Comparison

Model	Validation Accuracy
Self-improved 2D U-Net (30 epoch)	0.34522
Self-improved 2D U-Net (52 epoch)	0.57148
Self-designed 3D U-Net (30 epoch)	0.45129
SWIN U-Net	0.7740

5. Conclusion

After extensive analysis and training, we have gathered the key insights in the following section.

5.1. Learning Rate Scheduler

CosineScheduler can help the model escape local minima in the early stages of training by using a higher learning rate. In addition, by dynamically adjusting the learning rate, schedulers can make the training process less sensitive to the initial choice of learning rate.

5.2. Different loss distribution helps training

In the process of training Self-Improved Model, we introduce *penalty*. *Penalty* actually modifies the loss distribution. This helps our model escape from local minima and find better spot.

5.3. Batch Norm solves Gradient Vanish

In our basic U-Net, we encounter gradient vanishing problem which the model is not effectively updating. We use *batch norm layer* solve this problem. To be specific, we add ***nn.BatchNorm2D*** layers between each convolution layer pair.

5.4. Dimension of Convolution Layers matters

Self-Improved U-Net utilizes *2D conv. layer* while SWIN U-Net utilizes *3D conv. layer*, therefore SWIN U-Net will consider relationship between different layers while Self-Improved U-Net do not. Therefore, the accuracy of Self-Improved U-Net is lower than the accuracy of SWIN U-Net.

5.5. What's next

While we were able to evaluate our self-designed model and SwinUNETR model independently, if given more time, we would have tried to apply a collective classification across both models to get improved results. The memory limit on Google Colab and time it takes to train on a large dataset also continually prove to be a bottleneck for our progress, or we could attempt to train the models on the combined BraTS 2021 and 2020 datasets for better robustness against unforeseen data.

References

[1] S. Bakas, H. Akbari, A. Sotiras, M. Bilello, M. Rozycki, J.S. Kirby, et al. Advancing the cancer genome atlas glioma mri collections with expert segmentation labels and radiomic features. *Nature Scientific Data*, 4:170117, 2017. 1

[2] S. Bakas, M. Reyes, A. Jakab, S. Bauer, M. Rempfler, A. Crimi, et al. Identifying the best machine learning algorithms for brain tumor segmentation, progression assessment, and overall survival prediction in the brats challenge. *arXiv preprint*, 2018. 1

[3] CBICA. Brat dataset. Link to dataset, 2020. Accessed on 15 April 2024. 1, 2

[4] U.Baid et al. he rsna-asnr-miccai brats 2021 benchmark on brain tumor segmentation and radiogenomic classification, 2021. 1, 2, 4

[5] KidsWithTokens. Medical image segmentation with diffusion model. Link to github, 2023. Accessed on 19 April 2024. 2

[6] MONAI. Swinunetr model. Link to Google Colab, 2024. Accessed on 15 April 2024. 1, 2, 4, 5

[7] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *arXiv*, 2015. 1, 2

6. Appendix

All project codes and models can be viewed here: Google Drive: Project