3D Midical Mitochondria Segmentation

Yi Zhao

April 2024

1 Introduction

This study discusses the semantic segmentation implementations using *the Electron Microscopy Dataset* created by Aurélien Lucchi, Yunpeng Li, and Pascal Fua at EPFL.
This report includes the implementation of 3D semantic segmentation models and compares the results with those obtained from traditional 2D segmentation models. All of the source code can be found at this link

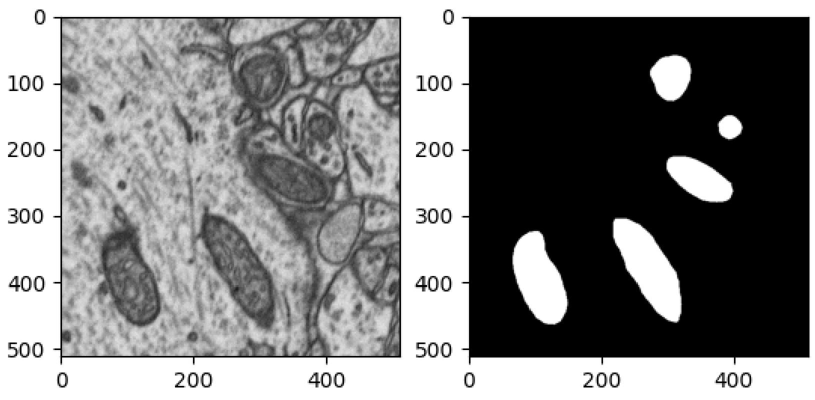


Figure 1.1: The sample image and mask slices extracted from the dataset

2 Preprocessing

2.1 Dataset Extraction & Data Cleaning

The dataset represented a 5x5x5 µm section taken from *the CA1 hippocampus region of the brain*, corresponding to a 1065x2048x1536 volume. It provided four TIFF files with annotated masks. The training dataset and testing dataset both had two files. One file contained the original image, and the other file contained the corresponding mask.
The training and testing data all have a shape of 165x1024x768, which means each image consists of 165 slices with 1024x768 voxels, respectively. In this report, I chose 128x128x128 as the crop shape, so each dataset has 48 slices.

2.2 Data Augmentation

I used simple augmentation techniques, such as horizontal/vertical flips, random brightness adjustments, etc. It seemed that these augmentations would lead to worse results during training, especially when using rotation and coarse dropout techniques. Due to a lack of training data, I did not include validation data in my work.

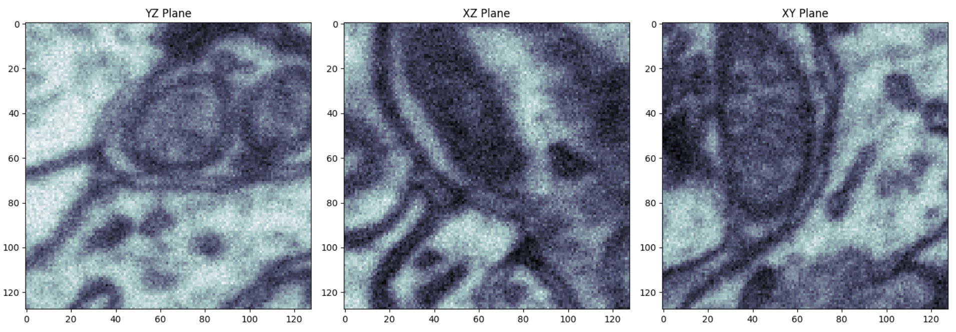


Figure 2.1: The sample 3d image slices extracted from the dataset

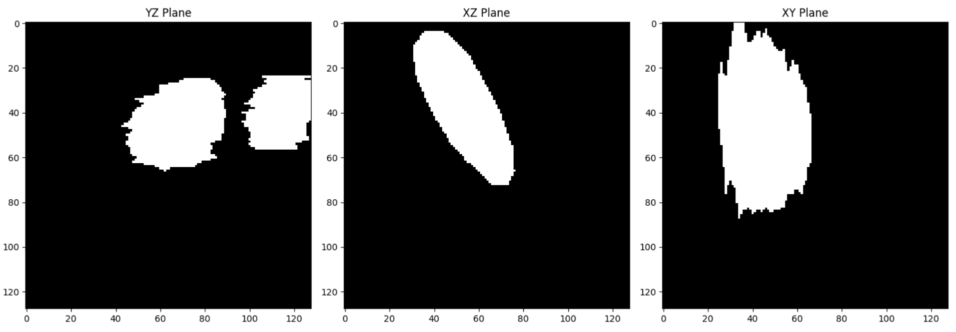


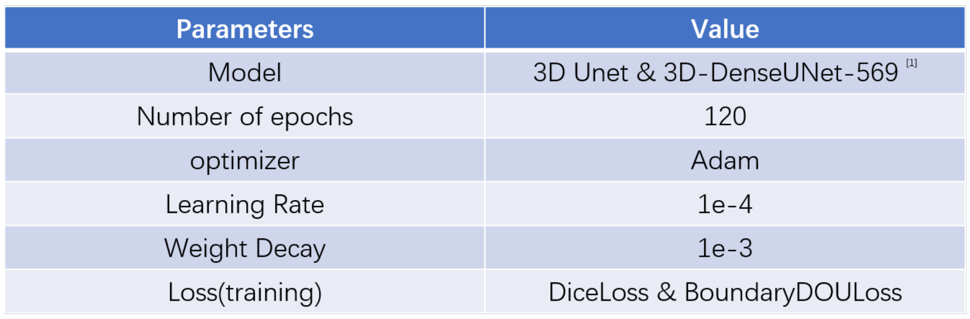
Figure 2.2: The corresponding 3d mask slices extracted from the dataset

3 Models & Metrics

3.1 Building the model

In this task, I developed a basic *3D UNet* to segment the 3D volumes. Instead of using the *MONAI U-Net* models, I redesigned the network. The network basically consisted of three modules: encoders, decoders, and the final layer.

The encoders consisted of numerous modules with convolutional layers followed by max-pooling layers, aiming to extract hierarchical features from the 3D volume. Upsampling layers and convolutional layers are essential components of the decoders. After passing through the final layer with the softmax function, the volume completed its segmentation. However, I used a unique loss function in my implementation, and I will discuss it in Section 3.2.



Parameters	Value
Model	3D Unet & 3D-DenseUNet-569 [1]
Number of epochs	120
optimizer	Adam
Learning Rate	1e-4
Weight Decay	1e-3
Loss(training)	DiceLoss & BoundaryDOULoss

Figure 3.1: model parameters

Besides, I utilized another 3D UNet architecture, the 3D-DenseUNet-569, to investigate whether adding dense modules would yield better results. *The 3D-DenseUNet-569* model adopted Depthwise Separable Convolution (DS-Conv) rather than tradional Convolution. Furthermore, it reduced the pooling layer by using a standard convolution with strides. From my perspective, decreasing GPU memory requirements can make it more accurate in segmentation tasks. Theoretically, the use of transfer learning may yield better results, but it did not seem to have the expected outcomes, **Refer to the results in part 4**.

3.2 BoundaryDoULoss VS DiceLoss

Initially, I trained the 3D UNet using the *DiceLoss* function with sample data. However, I found that the model showed little improvement in performance when more epochs were added. It also showed signs of overfitting due to the lack of data augmentation and the limited dataset size. Inspired by the shared paper, I used the *BoundaryDOULoss* in my training.

According to the shared paper, the BoundaryDOULoss is obtained by calculating the ratio of the difference set of prediction and ground truth to the union of the difference set and the partial intersection set. This method may make it easier to calculate the loss and speed up the training process. The BoundaryDOULoss is calculated as follows:

$$L_{\mathrm{DoU}} = \frac{G \cup P - G \cap P}{G \cup P - \alpha \cdot G \cap P} \tag{3.1}$$

DiceLoss is a loss function used to score ranges from 0 to 1, with a value of 1 indicating a perfect overlap between the predicted segmentation and the ground truth labels (masks). *DiceLoss* is calculated as follows:

$$DiceLoss\,(y, \overline{p}) = 1 - \frac{(2y\overline{p} + 1)}{(y + \overline{p} + 1)} \tag{3.2}$$

3.3 Other Metrics

Jaccard's Index/IoU and *DiceBCELoss*[1] are used for evaluating the model performance, with *DiceBCELoss* referring to:

$$DiceBCELoss = DiceLoss + BCELossWithLogits \tag{3.3}$$

And the *Jaccard's Index/IoU* refers to:

$$Jaccard(U, V) = \frac{|U \cap V|}{|U \cup V|} = \frac{|U \cap V|}{|U| + |V| - |U \cap V|} \tag{3.4}$$

[1]In this report, all of the models using DiceBCELoss as Loss function when testing models

4 Results

Name	Loss Type	Training Set			Testing Set		
		Loss[2]	Dice Score	Jaccard's Index/IoU	Loss[2]	Dice Score	Jaccard's Index/IoU
3D Unet	Dice	0.224	0.81	0.75	0.250	0.71	0.51
3D Unet	BoundaryDOU	0.140	0.85	0.80	0.184	0.72	0.65
3D-DenseUNet-569[1]	Dice	0.627	0.42	0.31	0.653	0.42	0.30
3D-DenseUNet-569[1]	BoundaryDOU	0.322	0.53	0.42	0.329	0.52	0.41

Figure 4.1: 3D UNet with different models and training loss

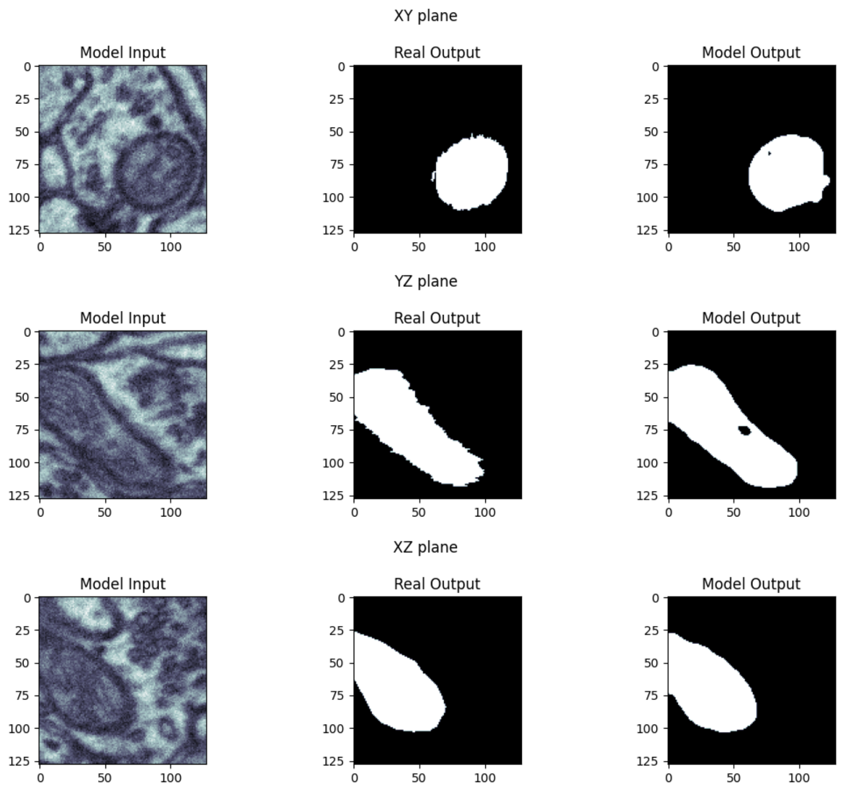


Figure 4.2: Sample Inference over testing set

Backbone[1]	Training Set			Testing Set		
	Loss[2]	Dice Score	Jaccard's Index/IoU	Loss[2]	Dice Score	Jaccard's Index/IoU
resnest14d	0.169	0.82	0.80	0.372	0.61	0.51
resnet34	0.229	0.66	0.61	0.348	0.49	0.42
resnest50d	0.113	0.84	0.78	0.288	0.60	0.47
resnet101	0.524	0.29	0.19	0.557	0.24	0.15
resnest200e	0.112	0.84	0.78	0.246	0.65	0.55

Figure 4.3: 2D UNet with different backbones

5 Conclusion

The use of the *BoundaryDOULoss* results in improved performance in both the *3D UNet* and *3D-DenseUNet-569 models*. It can provide quicker and more accurate results compared to using *DiceLoss*. The model converges quickly and has the ability to produce optimal results across the datasets. Furthermore, it reduces overfitting to some extent, which has greatly surprised me.
However, it may not be compelling since the dataset has limited data. Another phenomenon is that when adding rotation-related augmentation, the model failed to generalize, and the back-propagation process seemed very slow. This issue occurred with both models during training. From my perspective, I believe that the *3D UNet* may extract features more effectively, and adding rotation could potentially disrupt this process by introducing incorrect features.

Besides, I use *2D UNet* models with different backbones, such as *ResNet*, *ResNeSt*, etc. Each of them is loaded with pretrained weights from *ImageNet* during training. Some models performed even better than the **3D-DenseUNet-569** model. But these models all overfitted significantly, even though I only used horizontal and vertical flips for data augmentation.

This report only displayed two 3D models due to insufficient video memory, despite some codes running on Kaggle using the **P100**. For the reason that I am an entry-level student in 3D semantic segmentation, this report only uses one type of 3D slices to train models.
For future work, I'd like to remove small objects and apply customized methods to get better results! Also, I will utilize more advanced models(*eg. ViT*) once I have devices with greater video memory capacity!