Mapping Neural Circuits: Synaptic Segmentation

Blake Prall

A. Introduction

I am interested in the idea of mapping the human connectome through the use of deep-learning neural networks. Despite being the system by which we experience the world and the foundation of consciousness, we still know surprisingly little about the central nervous system due to its vast complexity. Connectomics aims to change that by directly mapping these circuits. There are multiple aspects that go into this challenging task, most regarding the segmentation of individual neurons in order to generate a volumetric representation of neural circuitry.

In my project, I aim to develop a neural network or system of networks that can both instance segment individual neurons as well as segment their synaptic clefts. I hypothesize that a U-Net3D-like architecture that predicts 3D bounding boxes of individual neurons from encoder feature maps, which then generates a neuron instance segmentation map through the decoder layers [7], alongside an FCN that predicts synapses [3] will best complete this task. However, despite my best efforts, I encountered difficulties in getting the desired results.

The problem at hand is not only of great interest but also holds significant implications. Understanding the structure and connectivity of neural circuits within the human brain has far-reaching consequences for neuroscience and cognitive science. The field of connectomics has the potential to provide crucial insights into the mechanisms underlying various neurological disorders and pave the way for the development of innovative treatments and interventions.

Unfortunately, I was unable to complete the proposed neural network architecture, which combines the instance segmentation of neurons and the segmentation of synaptic clefts, to work as intended. Yet, I still believe that this unified approach has the potential to advance the field of connectomics by enabling more accurate and comprehensive mapping of neural circuits.

Despite the setbacks, my work contributes to the broader understanding of the human connectome by emphasizing the importance of effectively segmenting individual synaptic connections. It underscores the need for continued research and exploration in this field to unlock new insights into brain function and advance our knowledge of the central nervous system.

B. Related Work

Most models for developing segmentation of both neural cells and synaptic clefts are done through an initial object detection phase, followed by a segmentation phase. There have been multiple models developed that are effective at completing this task, as well as multiple that were designed specifically for neuron and cleft segmentation.

B.1. PyTorch Connectomics

PyTorch Connectomics is an open-source python library designed specifically for connectomic research and analysis of electron microscopy data. There are many models utilized by this library, but the primary method for segmentation is through the use of an FPN, U-Net3D, and waterz [2].

This method works by first sending the input data through the FPN class to retrieve the ROIs. This data is then sent to a U-Net3D model, which creates an affinity map, essentially calculating whether two neighboring voxels belong to the same cell. Finally, the affinity maps generated by the U-Net model are then sent to waterz, which is another open-source tool designed to work with electron microscopy data. Unlike previously mentioned methods, waterz utilizes traditional watershed segmentation to generate masks. This method then uses the mask generated by waterz as the final instance segmentation mask.

To predict synaptic clefts, PyTorch Connectomics utilizes models which are similar to those that are used in affinity map generation when segmenting neurons. Meaning, they treat synaptic cleft detection as a semantic segmentation task and use encoder-decoder FCNs to achieve this.

B.2. Pixel-Wise Neural Cell Instance Segmentation

The method for pixel-wise neuron instance segmentation is quite similar to the previously mentioned method. Except, in this task, there is one encoder-decoder model used, where the encoder feature maps are used to generate ROIs which are then fed directly into the decoder that creates an instance segmentation mask [7].

The encoder is designed with a similar architecture to an SSD, meaning it is given set anchors of differing scale and shape, and the model tries to find the best ones [6]. The decoder then acts like a decoder found in a common FCN, where the decode layers perform ”up” convolutions, taking the previous layer, as well as skips from the encoder as input, generating a segmentation mask through deconvolutions [4].

In order for this model to work, the encoder of this model is trained first, using ground truth bounding boxes as the target. Once the encoder is trained, the decoder is then trained to generate an instance segmentation mask. The result is a single model that can perform pixel-wise instance segmentation.

B.3. Non-Isotropic Synapse Segmentation

There are millions of synapses, and their shapes are not uniform, so it is essential that methods that can detect and segment synapses of varying shapes are developed. Non-isotropic segmentation is a relatively difficult task compared to isotropic segmentation, so the advancements of models which can perform as effectively would greatly advance synapse detection in large volumetric datasets.

A solution to this was found by representing an isotropic field of view on non-isotropic data by altering a U-Net3D model [1]. Through doing this, rather than the trained model only being effective on data that is similar to that which it was trained on, the model is able to better generalize synapses in regions of the brain that differ vastly.

C. Method

The majority of my work was centered around the development of a voxel-based neural instance segmentation model. This is essentially a combination of two of the previously mentioned methods that have been used for neuron instance segmentation, being, PyTorch Connectomics [2], which utilized U-Net3D among other models, and a pixel-wise neural segmentation model [7].

Most of the time spent in the development of this model was attempting to be able to feed the 3D bounding boxes predicted by the encoder into the decoder of a U-Net3D-like model. Yet, I was not able to complete this task and instead shifted my focus to synaptic cleft segmentation

For this task, I elected to use an FCN based heavily off of the U-Net model [5]. I chose to use an FCN, as I planned on completing this task via semantic segmentation, and these types of models have been shown to be effective in performing this task [3].

C.1. U-Net

The U-Net model 1 works as an encoder-decoder FCN that passes ”skips from the encoder to the decoder in order to preserve image information. The encoder essentially ”down-samples” the input, extracting more prominent features. The decoder does the opposite and ”up-samples,” using the extracted features and ”skips” from the encoder to generate a final mask [5].

The ”skips” passed from the encoder to the decoder are actually the encoder feature maps, and these maps are concatenated with the previous decoder map in order to give the decoder layer convolutions more data and context when generating the decode feature maps.

This model can take also inputs of varying sizes and can be easily converted to process 3-dimensional data dimensions simply by changing the dimensions of the convolution and normalization functions.

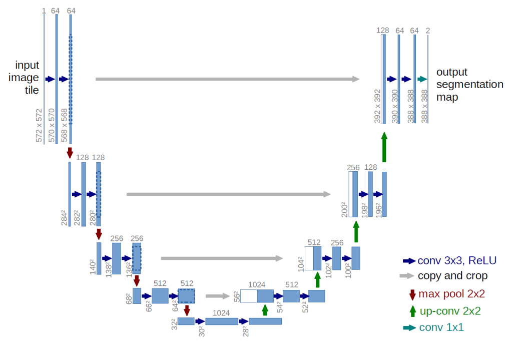

Figure 1. U-net architecture. Each blue box corresponds to a multi-channel feature map. The number of channels is denoted at the top of the box. The x-y size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote different operations. [5]

C.2. Loss Functions

I used three loss functions with differing weights to train my model, giving each a different weight before being sent to the optimizer. Binary-Cross-Entropy loss (1) measures the dissimilarity between the predicted probabilities and the true binary labels. It encourages the model to assign high probabilities to the correct class and low probabilities to the incorrect class. This aids the model in predicting whether or not a particular pixel (voxel) is background or not. For this reason, I set the weight of this loss function to be very low, as our background labels vastly outnumber the cleft labels, meaning BCE would be encouraged to predict nearly everything as being background.

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)] \quad (1)$$

The next loss function I used was Dice loss (2), which is a widely used loss function in image segmentation tasks that measures the dissimilarity between the predicted segmentation mask and the ground truth mask. It is derived from the Dice coefficient, which quantifies the overlap between two sets. The Dice coefficient is calculated as twice the intersection of the predicted and true masks divided by the sum of their areas. The Dice loss is particularly useful in synaptic cleft segmentation, as synaptic clefts are extremely small masks relative to the total size of the data, and Dice Loss is known for producing accurate segmentation results even when dealing with imbalanced data. it is because of this that Dice Loss was the most heavily weighted loss function while training the model.

$$\mathcal{L}_{\text{Dice}} = 1 - \frac{2 \cdot \text{TP}}{2 \cdot \text{TP} + \text{FP} + \text{FN}} \quad (2)$$

IoU Loss (3) is similar to Dice Loss, in that it measures the overlap between the predicted and true segmentation

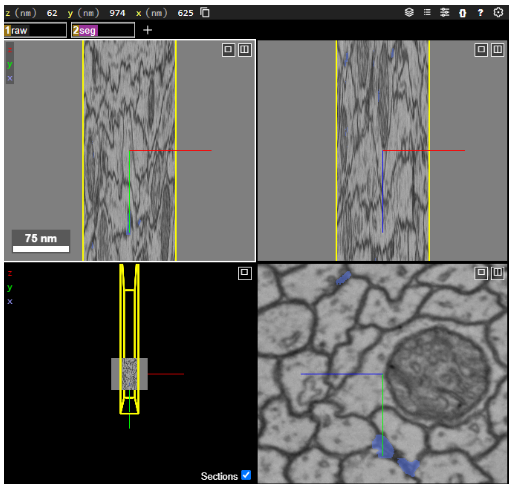

Figure 2. A view of a small part of the generated synapse segmentation mask in Neuroglancer.

masks. IoU does this by calculating the ratio of their intersection over their union. The value produced by calculating intersection over union ranges from 0 to 1, with 1 indicating a perfect match. IoU loss is then defined as 1 minus the IoU value and is minimized during training to encourage better alignment between predicted and true masks. IoU Loss, like Dice Loss, is effective for handling class imbalance and evaluating segmentation accuracy.

$$\mathcal{L}_{\text{IoU}} = 1 - \frac{\text{Area of Intersection}}{\text{Area of Union}} = 1 - \frac{\text{TP}}{\text{TP + FP + FN}} \quad (3)$$

C.3. Mask Generation and Display

To generate the synaptic cleft mask, I implemented the sliding window technique for three dimensions. Essentially, using three nested for loops, the model iterates along the entire input volume, using a patch that is the same size as the subvolumes the model was trained on, and a stride that is equivalent to the size of each dimension of the patch. Each of these individual patch masks is then added to a larger segmentation mask, giving a final mask once the entire input volume has been iterated upon.

This mask and the raw input data can then be fed into a library called Neuroglancer for display 2. Neuroglancer is a WebGL-based API that allows for volumetric visualization of the segmentation mask alongside the raw input data.

D. Experiments

I trained my model on the ”Cropped Version” of ”Dataset A” of the CREMI challenge. This dataset is a volume of 125 1250x1250 EM images, their instance segmentation mask, and a synaptic cleft mask. I pre-process each of these inputs separately.

For the raw input data, I first use histogram equalization in order to enhance the contrast of the image. The purpose of this is to improve the visual appearance by stretching the intensity range across the entire histogram. After initial equalization, adaptive Gaussian thresholding is then performed so as to help separate the image into different ”regions.” This produces an image with very low contrast, so histogram equalization is performed once again, and the raw input is set to the range [0,1]. To pre-process the synaptic cleft mask, I simply set the background values to 0 and the mask values to 1.

I then extract subvolumes of uniform shape from the mask and input volume to be used in the analysis by my model. I found that, for synaptic segmentation, smaller subvolumes not only trained faster but were also far more accurate. This is likely because the synapses are extremely small, and giving the model too much context likely hinders feature extraction. I also found that excluding negative subsamples (samples without synapses) essentially completely prevented any sort of meaningful training from occurring, and the final mask had predicted synapses everywhere. So, I ended up using subsamples of size (32,64,64).

After the subvolumes are extracted, the labels and input data are loaded together into my custom dataloader. In the loader, I make the list of subvolumes 5 times as long, as when an item is called, there is a probability of $\frac{1}{2}$ that the data will be flipped in any given direction, and a probability of $\frac{1}{4}$ that the image will be rotated either 0, 90, 180, or 270 degrees along the z-axis. This allows my relatively small dataset to become quite large, permitting more training while preventing overfitting. With the sample size (32,64,64), I had 4,060 samples to use in training, with $\frac{1}{5}$ of the samples still held for validation and testing.

With this much data, I was able to successfully train my model in 10 epochs with a batch size of 10 at a learning rate of 0.005. This training took about 2 hours with GPU acceleration (using a GTX 1070). This got my binary test accuracy to 0.986855 and my validation loss down to 0.009402, signifying successful training.

E. Conclusions

Through the development of my project, I realized the vast importance that data plays in computer vision. It seems to me that data and data preparation is the most important aspect of developing a good system. It is also evident to me that the model I am using for synapse segmentation could be improved, particularly by implementing bounding box prediction prior to synaptic cleft prediction, but I also think that the implementation of my original goal could aid in the detection of synapses as well. Although I was not successful in creating my ”all-in-one” model, I still think it is possible and has the potential to advance connectomics.

F. Contribution

- Blake Prall - All Sections and Code

https://github.com/bprall/CREMI-Neuron-Cleft-Segmentation

References

[1] Larissa Heinrich, Jan Funke, Constantin Pape, Juan Nunez-Iglesias, and Stephan Saalfeld. Synaptic cleft segmentation in non-isotropic volume electron microscopy of the complete drosophila brain. In Alejandro F. Frangi, Julia A. Schnabel, Christos Davatzikos, Carlos Alberola-López, and Gabor Fichtinger, editors, *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*, pages 317–325, Cham, 2018. Springer International Publishing. 2

[2] Zudi Lin, Donglai Wei, Jeff Lichtman, and Hanspeter Pfister. Pytorch connectomics: A scalable and flexible segmentation framework for em connectomics. 12 2021. 1, 2

[3] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015. 1, 2

[4] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. volume abs/1505.04366, 2015. 1

[5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. volume 9351, pages 234–241, 10 2015. 2

[6] Jingru Yi, Pengxiang Wu, Daniel Hoeppner, and Dimitris Metaxas. Fast neural cell detection using light-weight ssd neural network. pages 860–864, 07 2017. 1

[7] Jingru Yi, Pengxiang Wu, Daniel Hoeppner, and Dimitris Metaxas. Pixel-wise neural cell instance segmentation. pages 373–377, 04 2018. 1, 2