Semester Project Report

Promoting Connectivity of Linear Structures in 3D Microscopy Images

Taha Zakariya

Data Science

MA2

SCIPER : 288526

12 credit course

Professor: Pascal Fua

Supervisor: Doruk Oner

June 10, 2022

Contents

List of Figures

Abstract

As in many areas of computer vision, deep networks now deliver state-of-the-art results for delineation tasks, such as finding axons and dendrites in 3D light microscopy images. Most of the existing approaches rely on convolutional networks to extract from images binary masks denoting which voxels belong to neurites and which do not. Unfortunately, they do not guarantee that the connectivity of the produced masks corresponds to that of the real neurite network. This is because these methods are trained to minimize losses, such as cross-entropy and mean squared error, that do not explicitly enforce topological consistency. When the annotations do not perfectly coincide with the imaged structures, networks trained with the per-voxel losses produce results plagued by topological errors, such as interruptions and false positive connections.

Thus, the goal of this project is to experiment a new approach with recent brain data in order to promote connectivity of neurons in 3D predictions. We first test a basic Unet trained with one brain data that we use to experiment, then we expand our training to more brains, testing multiple loss functions that are meant to promote connectivity.

1 Introduction

First, given one brain with annotated neurons, a network was trained to segment other brains. Later, we received new brain data that we will use to first experiment the initial network as in sections 3, and see how it performs in Vaa3d's interface, as you can see in section 4. Then, given that some brains were manually annotated, we will use them to train new networks and do some new experiments as in section 5

In section 4, we will present clear visuals of how predictions perform and how precise they are compared to unprocessed data.

Then, in the section 5, we try different types of network architectures (Unet, ResUnet, Resnet encoder with Ce-net decoder) to see which performs best. After choosing the best model, we will test the performance of our model and data by trying multiple loss functions which results you will most likely see during the final presentation.

2 Distance maps

2.1 Dealing with huge but sparse data

The first problem we have to deal with when working with biomedical data is the huge amount of data that comes from it being in 3D and in very high resolution, so it takes hours to train and process. Nevertheless, we're only interested in neurons which don't occupy most of the brain, but are rather thin, connected and only live in some places of the brain. It's in fact very sparse, thus we divided the cube in 3D cubes, then we deal only with the cubes that contain these neurons.

We do that by starting with a cube or cubes which contain a neuron (it's the input of the algorithm), then we explore the neighboring cubes of this previous cube, and if they contain themselves a part of the neuron, we put them in the queue which cubes we will explore later on, in a kind of breadth-first search (BFS). At the end, when the queue won't contain anymore elements, we would have explored all the potential cubes that may contain a neuron, and segmented those who do.

2.2 Model

While going through data and keeping track of all the neighbors, at each step we segment the cubes that contain neurons.

The Unet is a convolutional network architecture for fast and precise segmentation of images. It has outperformed most methods when it comes to segmentation given that by nature, it's built to extract the most important features needed for this task. In particular, it works in general very well for biomedical 3D data as in [1].

For that reason, because we want to capture the distance from neurons, our baseline model was a Unet with 5 levels (4 levels encoder, then bridge, then 4 levels decoder).

At each level in the encoder (1 level = 1 down block), we start by max-pooling. Then we perform a 3D convolution, followed by a batch normalization and finally we apply a Rectified Linear Unit (ReLU). We do this twice, then we add a dropout layer at the end of the block. We also keep track of the outputs that we will use in the up-blocks.

Concerning the bridge, it's a block similar to a down block that we explained above, meaning that we max-pool, then we do a convolution, we batch normalize, we apply a ReLU (we do these twice), and then we do a dropout at the end.
Finally, at each level of the decoder (1 level = 1 up-block), we start by up-sampling through a transpose convolution where we reduce the number of channels by half, then we we concatenate with the input of the equivalent down-block. Then we do this twice: convolution, then batch normalization and finally ReLU. At the end of each block we also apply a dropout.
Then at the end of this network, we apply a convolution to get the initial number of channels. trained with brained 175 only to segment cube by cube

2.3 Visualization

This initial model was trained with 1 brain only (let's say brain 175), and we applied the network to compute the distance maps of the other brains. Below you can see the distance maps of the projection of a cube of brains 244 and 225 in Fig.1, and the down-scaled results over the whole brain of brains 175, 225 and 250 in Figs. 2, 3, 4.

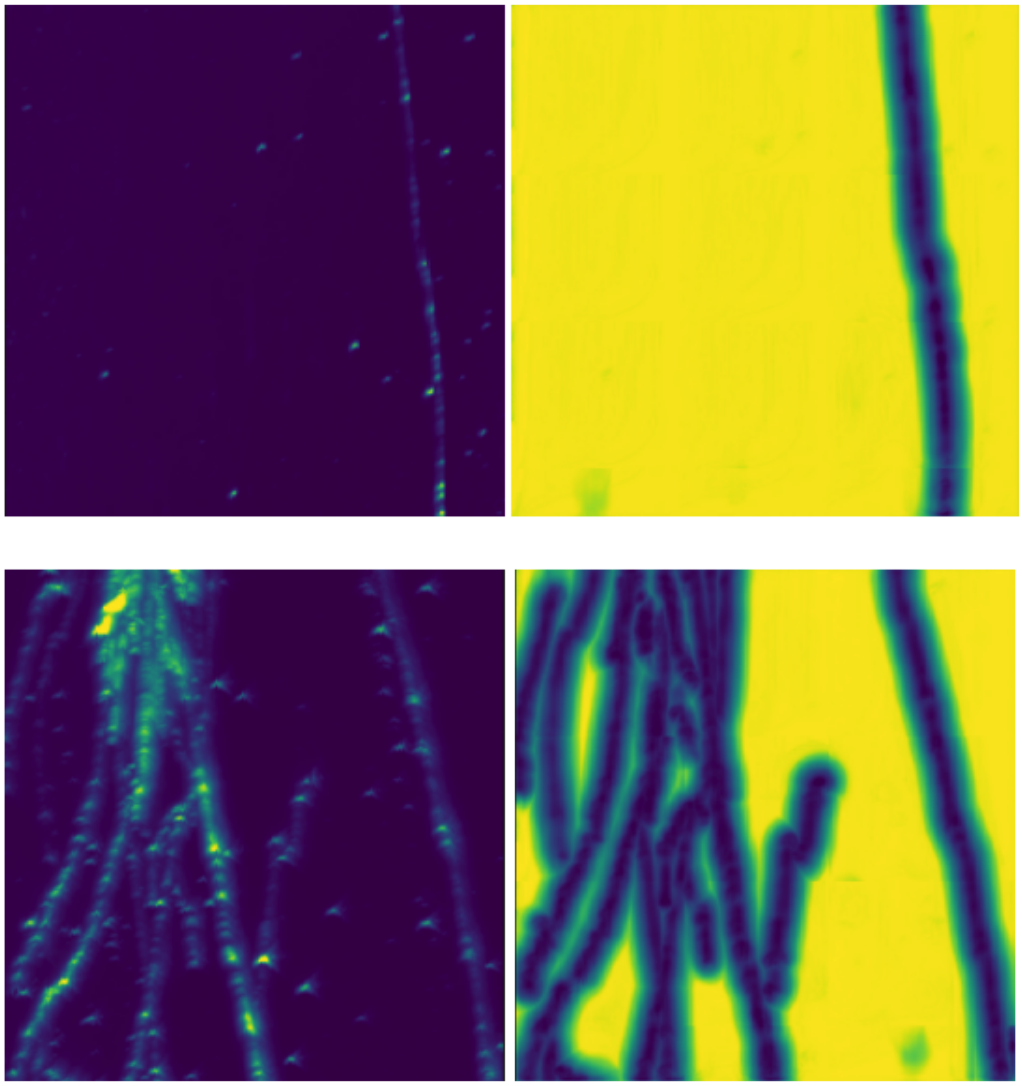

Figure 1: Left: Projection of one cube of the brains, Right: Distance maps, Up: Brain 244, Below: Brain 225

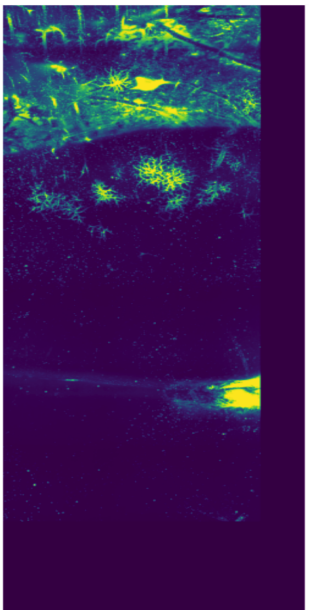

Figure 2: Down-scaled segmentation of neurons of brain 175

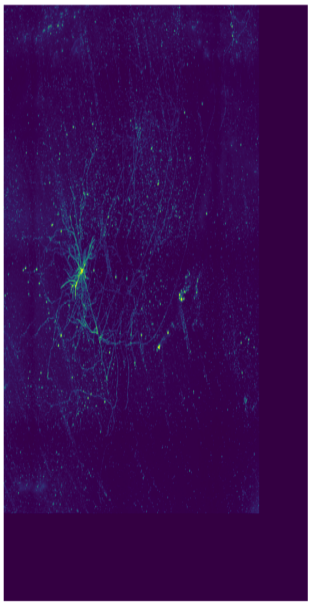

Figure 3: Down-scaled segmentation of neurons of brain 225

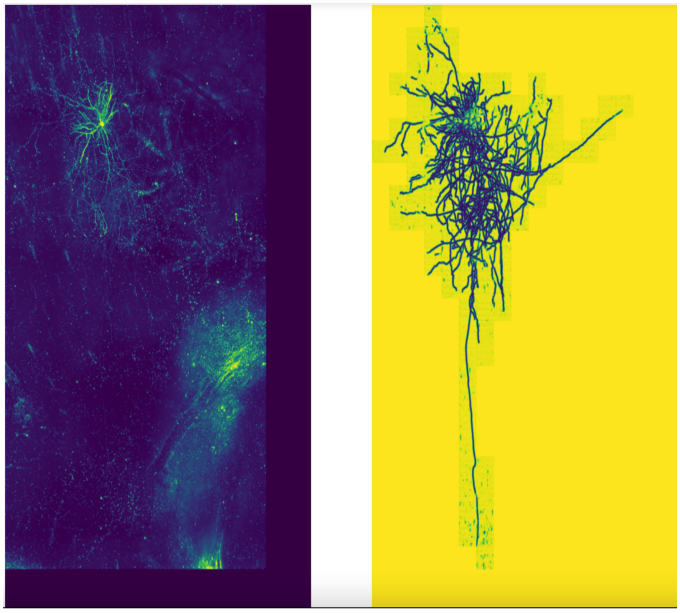

Figure 4: Down-scaled segmentation of neurons of brain 250

3 Segmentation

3.1 Baseline and prediction overlay

Now that we have a network that gives quite good distance maps, we move to the main task which is segmenting the neurons of the cubes. In order to do that we must overlay the distance maps with the base images in a way that distinguishes the neurons in comparison with the other cells.

An important constraint was that the data we produce should be compatible with Vaa3D that the lab we're working with will use, which added some constraint on the data. For example data type should be int16, and values range should vary between 0 and 65535, so scaling in this range was important. Finally we had a decision to make: choosing between RGB and Gray-scale.

3.2 Adapting the prediction to vaa3d: RGB vs Grayscale

Initially, the data we have is in gray-scale. However, given that overlaying is much easier and looks much nicer with RGB, we first tried to do it with RGB and make it work on Vaa3D even if we knew some data would stay gray-scale while other images will become RGB. We thought it might work given that Vaa3d analyses cubes separately when we're on high resolution. However, it didn't work and we decided to go with gray-scale after seeing that the segmentation was more than acceptable as you'll see bellow.

For the RGB segmentation, we put the prediction on the green channels and duplicated the gray-scale channel.

For the gray-scale segmentation, we initially normalized and added the prediction multiplied by a factor then re-scaled everything back.

In figures 5,6 , you can see the difference between the initial image, the RGB-segmented one and the grayscale-segmented. Note that this prediction is already optimized, but we will talk about how we optimized it on section 3.3

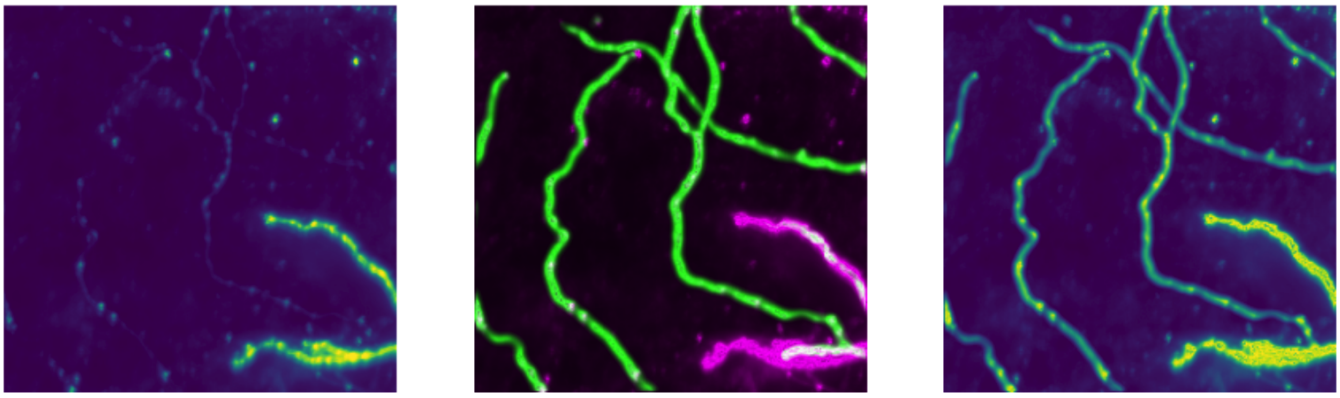

Figure 5: RGB vs Grayscale segmentation comparison for a cube of brain 175. Left: initial cube image, middle: RGB, right: gray-scale

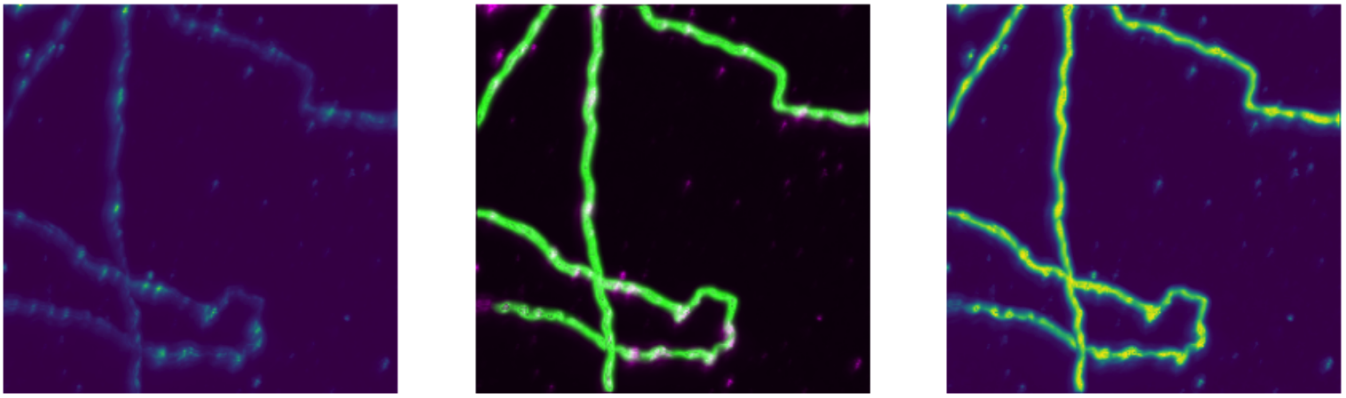

Figure 6: RGB vs Grayscale segmentation comparison for a cube of brain 225. Left: initial cube image, middle: RGB, right: gray-scale

3.3 Optimizing the prediction

In order to optimize our prediction that took a larger spectrum initially, we removed the large values of the distance maps by clipping them. Then in order to make it even sharper, we powered the values before normalizing again, which gave much more importance to the values nearer to the neuron, and thus brighter for the image/prediction overlay near the neuron as you can see in Fig. 7.

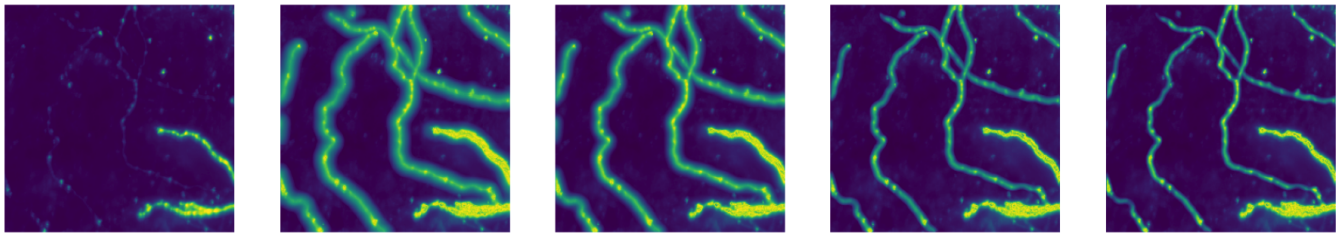

Figure 7: Respectively from left to right: initial image, gray-scale segmentation with no optimization, gray-scale segmentation with no optimization with clipping values to 10, gray-scale segmentation with no optimization with clipping values to 6, gray-scale segmentation with no optimization with clipping values to 6 and powering to 2.

Thus, here's the final result on brains 175 and 225:

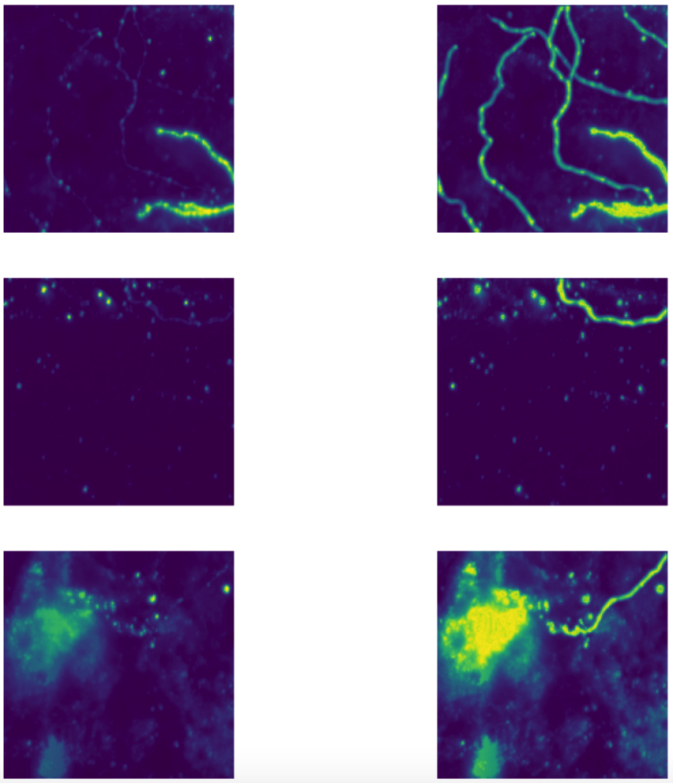

Figure 8: Initial and overlayed prediction of a cube of brain 175

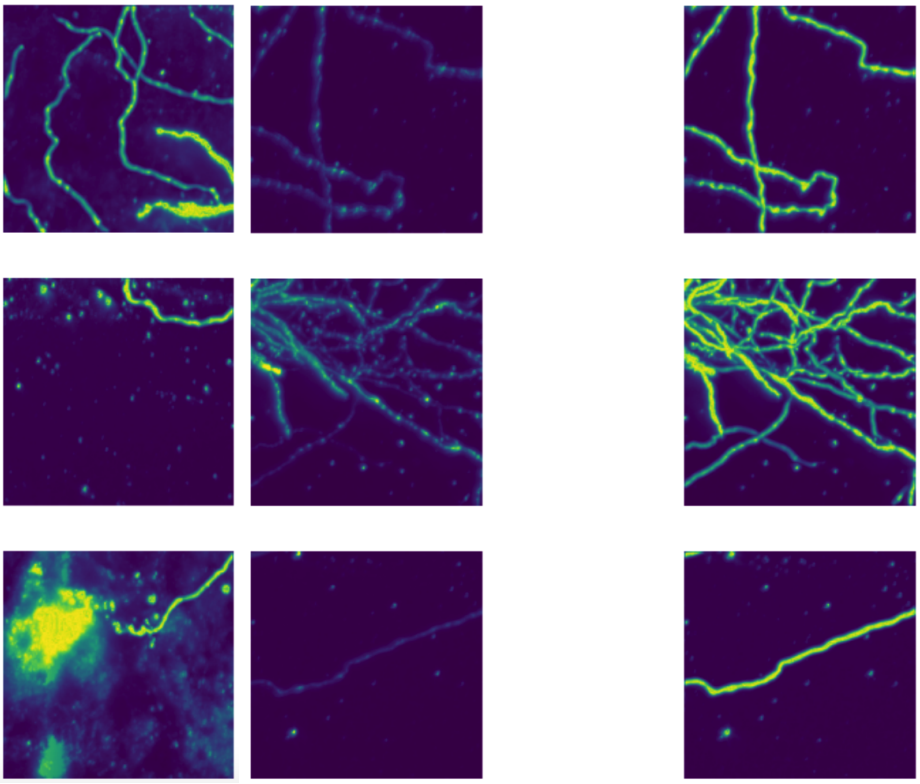

Figure 9: Initial and overlayed prediction of a cube of brain 225

4 Vaa3d Performance

After segmenting all these cubes of the needed brains, it was time to see how it performs on Vaa3d and if this segmentation really improves Vaa3d neurons detection.
A video is available at `https://drive.google.com/file/d/19wERUcF7qrk3IWhBHu_Pi_uWy3MYmZYu/view?usp=sharing` showing how much it improves the detection, but for this report I'll show you some images.

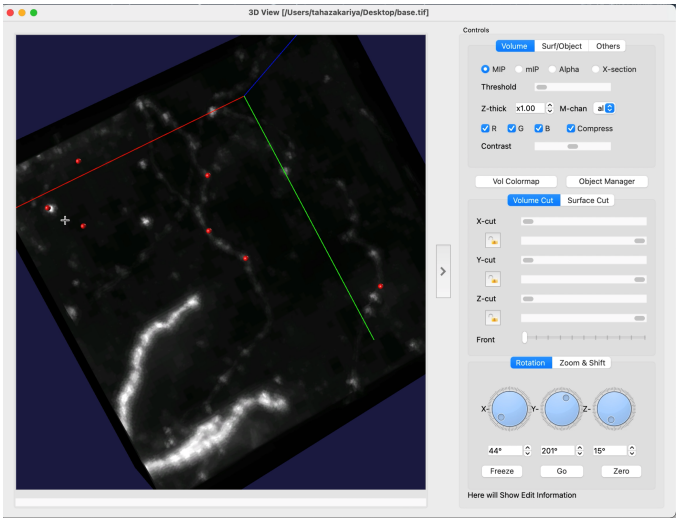

Figure 10: Neuron detection before overlaying the segmentation

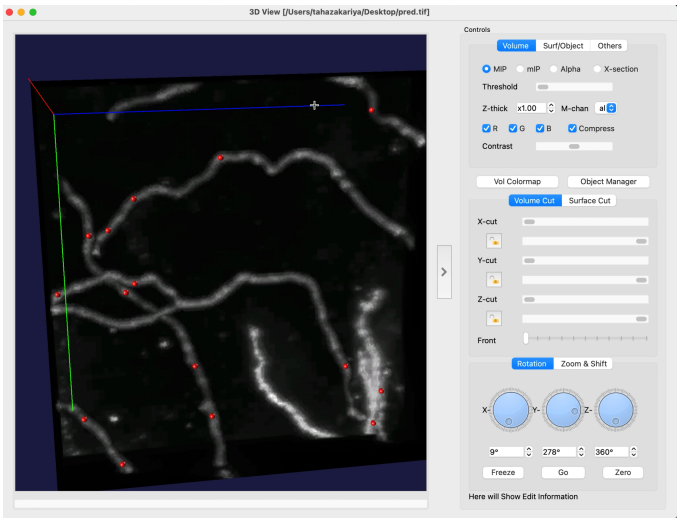

Figure 11: Neuron detection after overlaying the segmentation

You can clearly see that on the left, some red points are totally random, not on any "white line" which represent neurons, this is due to trying to place the markers from difficult angles, which we need to make sure works given the data is 3D. However, on the right screen, you can see that Vaa3d detects perfectly where the neuron is when placing the markers from any position or angle.

5 Retraining

After getting these results from a network trained with only one brain (brain 175), we received new annotated brains data. Thus, we decided to make a better network for neurons segmentation.
This will be done through multiple areas: first we include the new data as explained in section 5.1, then we test new networks architectures as in section 5.2 and finally test other loss functions in section 5.3.

5.1 Rendering and new DataSet

Before implementing a Dataset that takes care of the new brains, we needed to get all the necessary data which include the cubes, the distance maps and the labels (which are the binary rendered cubes). For the rendering we had a function that was used for brain 175, and for the distance map we used the distance transform of scipy's library.
Also, it's important to note that we don't use all cubes, but only those whose rendering contains a minimum number of pixel neurons, that we have set to 100. We do this for computational time purpose and efficiency as there's no need to take them into account. We have chosen 100 as a threshold because it removed enough and not too much data, as you can see in the histograms below:

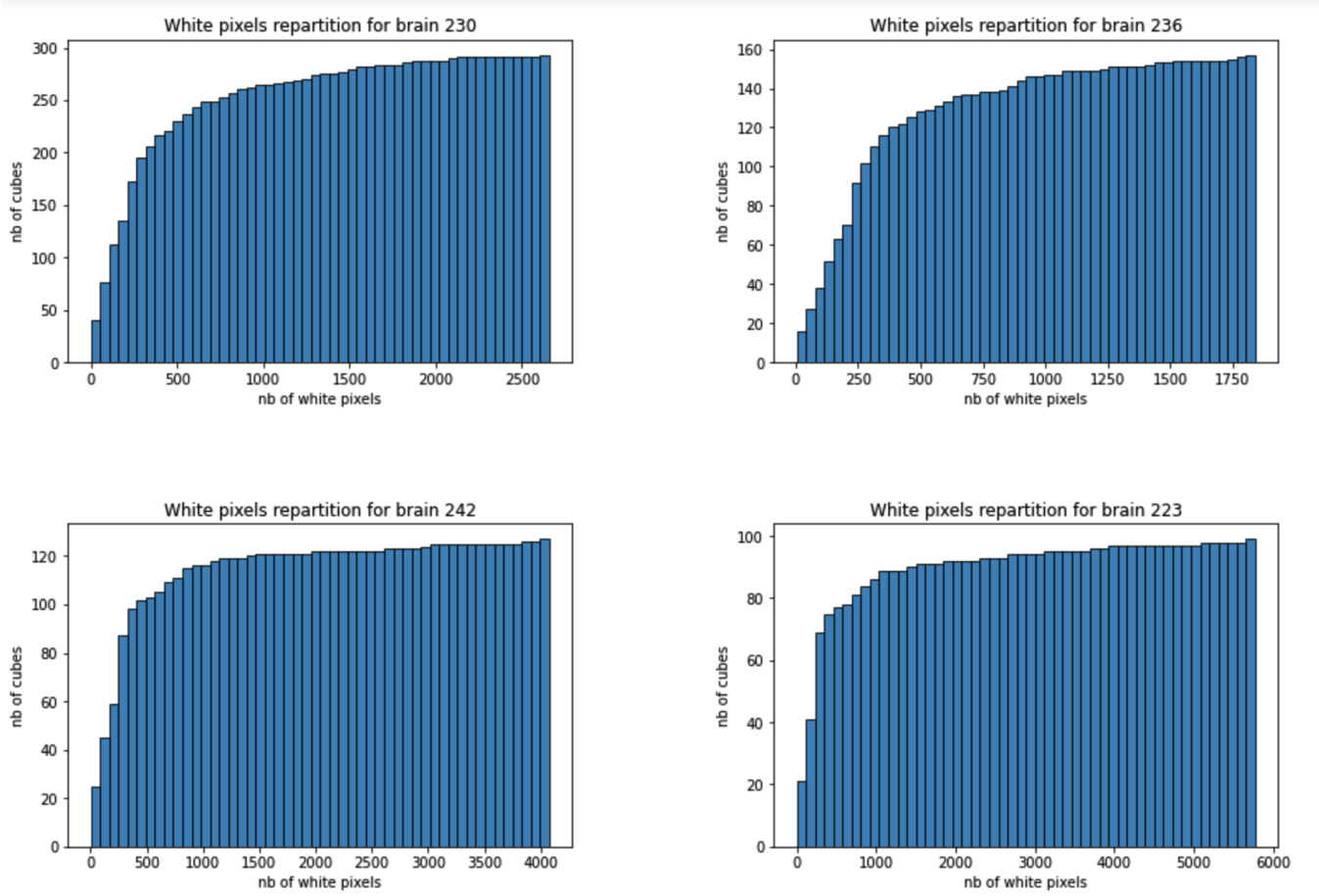

Figure 12: Cumulative repartition of white pixels over the cubes of each used brain (noted in the title)

So, for each cube, we save 3 images: the cube itself, the rendering and the distance map, as in Fig. 13

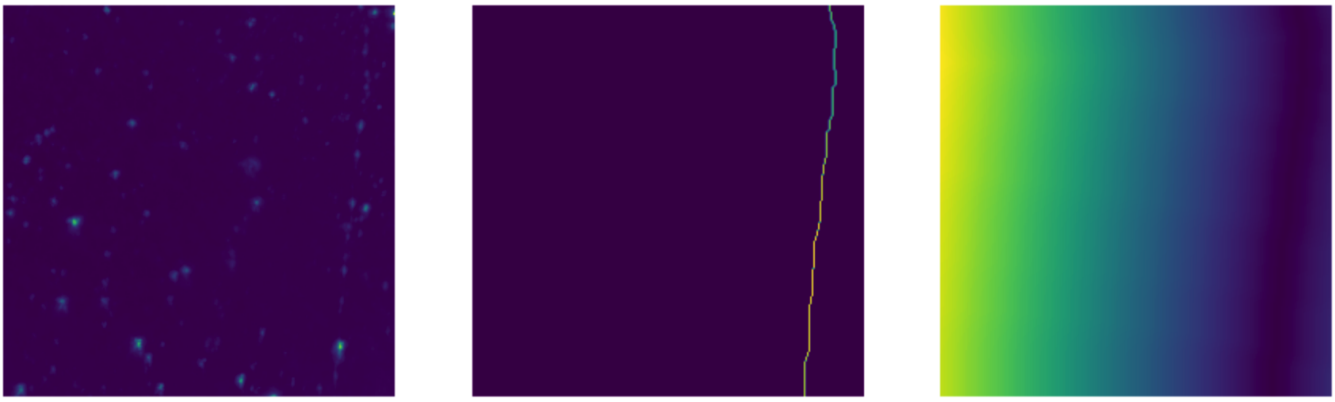

Figure 13: Input of the Dataset, respectively the cube, the rendering and the distance map, all down-scaled for visualization matter

However, we don't use this data directly as it is, because as you may have noticed, the distance maps don't look very informative at this point. For that purpose, we pre-process them in a similar way of section 3.3, we clip the values to a maximum of 10, which at the end gives this data that the network will take as an input:

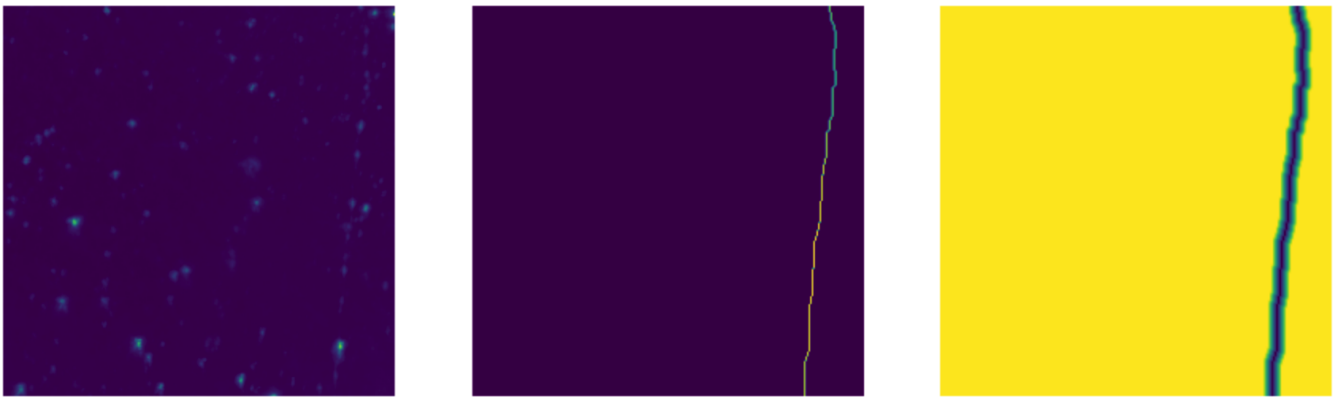

Figure 14: Output of the Dataset, respectively the cube, the rendering and the distance map clipped to 10, all down-scaled for visualization matter

5.2 Other networks

5.2.1 Our network

We have already presented our initial network in section 2.2. Nevertheless, it didn't work well, so we have tried new combinations of it with, like changing the the number of channels, number of convolutions... but it still gave a bad output like these:

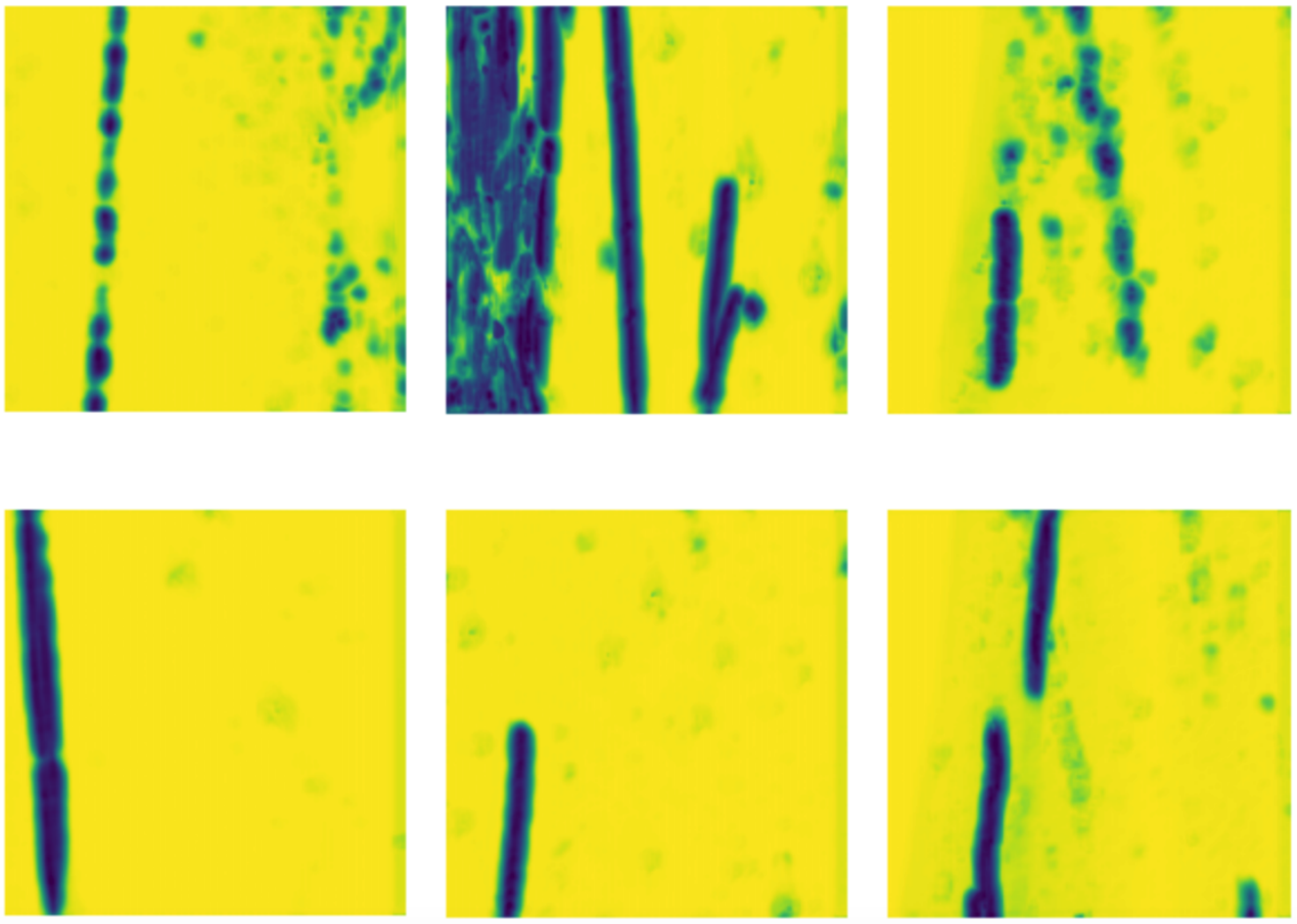

Figure 15: Output of the network for some cubes

Thus we decided to explore new networks and loss functions, which we'll talk about below.

5.2.2 Previous works

There exist multiple previous works that do medical image segmentation, but none that do exactly what we need, thus we'll explore multiple paths.

A) MedicalNet MED3D: transfer learning for 3d medical image analysis

Many studies have shown that the performance on deep learning is significantly affected by volume of training data. The MedicalNet project [2] aggregated the dataset with diverse modalities, target organs, and pathologies to to build relatively large datasets. Based on this dataset, a series of 3D-ResNet pre-trained models and corresponding transfer-learning training codes were provided.
In a nutshell, MedicalNet proposes a heterogeneous Med3D network to be used for 3D multi-domain medical data tasks, which can extract general 3D features even in the case of large differences of data domain distribution. In other words, they trained multiple heterogeneous models on large amount of 3D data, which can be used as the backbone pre-trained model to boost other tasks with insufficient training data other 3D medical image tasks that can then fine-tune the parameters.

This training has been done first through collecting several publicly available 3D segmentation datasets, then training through a family of ResNet models as the basic structure of the encoder and do minor modifications to allow the network to train with 3D medical data.
They proved that taking their model as a backbone improves performance performance of medical tasks immensely: between 5% and 30% improvement for lung segmentation and pulmonary nodule classification.

B) CE-Net: Context Encoder Network for 2D Medical Image Segmentation

As said in the beginning, U-net based approaches are very widely used when it comes to mdecial data segmentation. However, the consecutive pooling and strided convolutional operations lead to the loss of some spatial information. In CE-Net's paper [3], they propose a context encoder network (referred to as CE-Net) to capture more high-level information and preserve

spatial information for 2D medical image segmentation. CE-Net mainly contains three major components: a feature encoder module, a context extractor and a feature decoder module. An important note is that they use pretrained ResNet block as the fixed feature extractor.

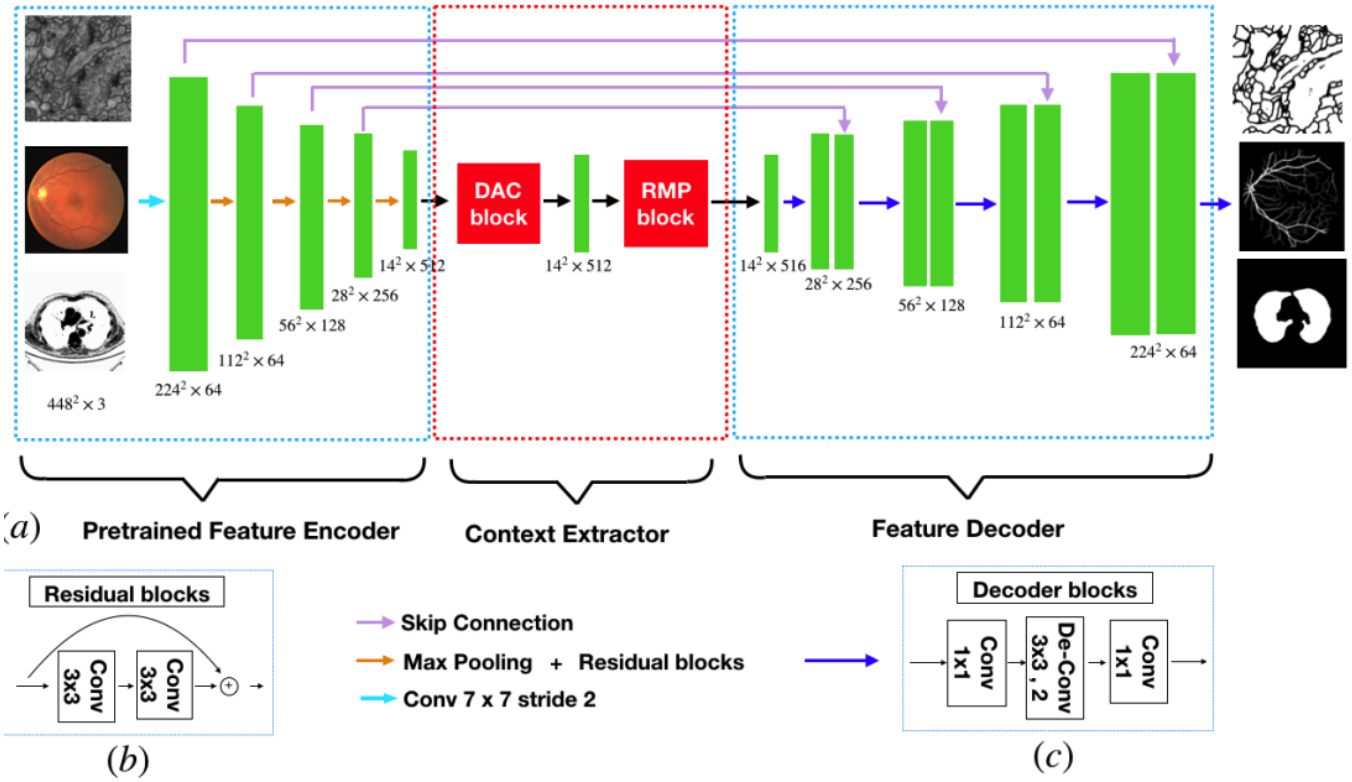

Figure 16: Illustration of the CE-Net. Firstly, the images are fed into a feature encoder module, where the ResNet-34 block pretrained from ImageNet is used to replace the original U-Net encoder block. The context extractor is proposed to generate more high-level semantic feature maps. It contains a dense atrous convolution (DAC) block and a residual multi-kernel pooling (RMP) block. Finally, the extracted features are fed into the feature decoder module. They adopt a decoder block to enlarge the feature size, replacing the original up-sampling operation. The decoder block contains 1×1 convolution and 3×3 deconvolution operations. Based on skip connection and the decoder block, they obtain the mask as the segmentation prediction map.

The Feature Encoder Module is what makes their module very efficient in segmentation, in particular the fact that they're using a ResNet a feature extractor.

C) ResUnet++ and Road Extraction by Deep Residual U-Net

These last two previous works [4], [5] argue that a combination of a ResNet and a U-Net outperforms both models (which are the state-of-the-art models of the domain) if used separately. They argue that the benefits of this model is two-fold: first, residual units ease training of deep networks. Second, the rich skip connections within the network could facilitate information propagation, allowing to design networks with fewer parameters however better performance. They propose the following architectures:

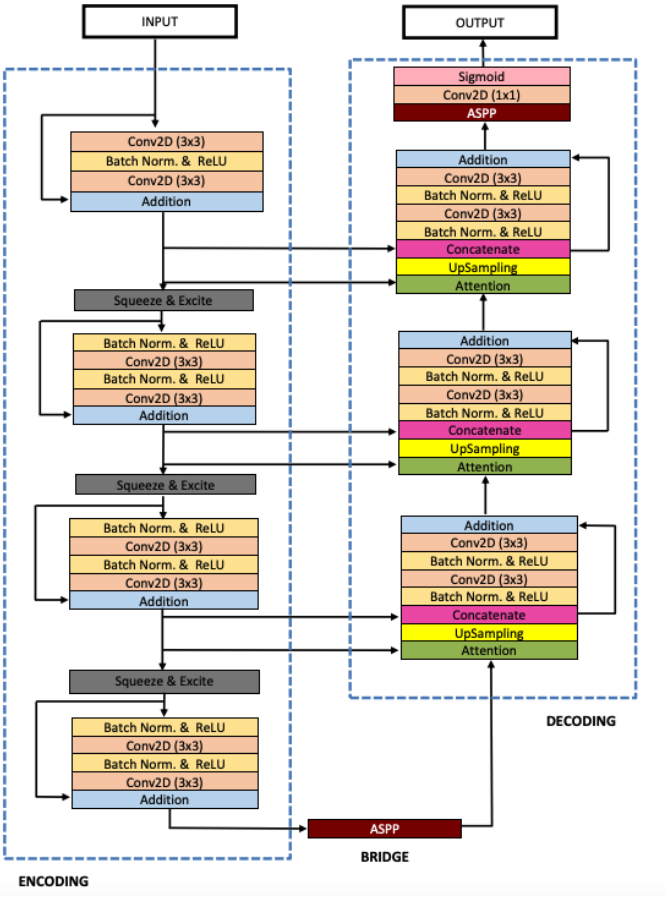

Figure 17: Block diagram of the proposed ResUNet++ architecture.

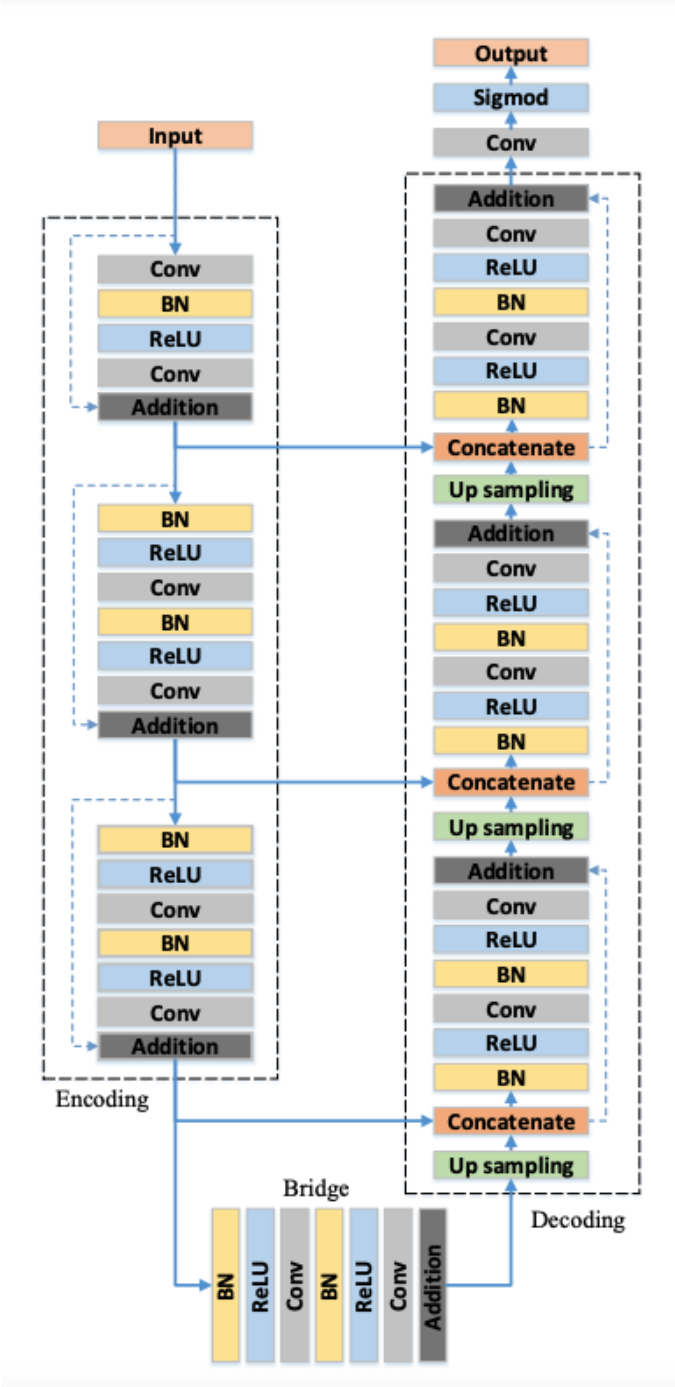

Figure 18: The architecture of Deep Residual Unet.

As you can see, in both cases they use a ResNet as a feature extractor and then a Unet for decoding.

In semantic segmentation, to get a finer result, it is very important to use low level details while

retaining high level semantic information [1], [6]. For this matter, The deep residual unit makes the deep network easy to train and the skip connection within the networks helps to propagate information without degradation, improving the design of the neural network by decreasing the parameters along with comparable performance or boost in performance on semantic segmentation task [4].

5.2.3 What we'll be doing

The common ground between all these previous works is that they use a ResNet as a feature extractor, and some of them provide a pre-trained version adapted for biomedical segmentation. Thus in the few upcoming days, we'll be testing 2 networks:

- ResUnet: we'll modify our network so that we have a new option for the encoder which is the resnet as a feature extractor.
- CE-Net: We'll take the provided CE-Net code that's used for 2D Segmentation and we'll adapt it to our data to see if we can leverage it into a good biomedical 3D neuron segmentation network.

5.3 Loss functions

Until now, we used MSE loss to train the network. However, once we get a good network, we'll be testing new loss functions.
First, we're going to test what we call MSE-GaussSnake-wGrad [7], which is a loss based on snakes algorithm [8] [9], which aims to minimize energy functions generated by deep networks, which enables end-to-end learning by adjusting the annotations during training.
For the last two loss functions we will use, they've been designed to enforce continuity of 2D linear structures. The building block of these loss functions is L_{TOPO} from [10]. From this loss, we first implement an intermediary loss L_{conn} which is :

$$L_{\text{conn}}(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{i \in \{x,y,z\}} L_{\text{TOPO}}\left(\mathbf{y}^i, \hat{\mathbf{y}}^i\right) \tag{1}$$

Then we can get the two losses we'll be testing, L_{3D} and L_{2D}:

$$L_{3D}(\mathbf{y}, \hat{\mathbf{y}}) = L_{\text{MSE}}(\mathbf{y}, \hat{\mathbf{y}}) + \alpha L_{\text{conn}}(\mathbf{y}, \hat{\mathbf{y}}) \tag{2}$$

$$L_{2D}\left(\mathbf{y}, \hat{\mathbf{y}}_x, \hat{\mathbf{y}}_y, \hat{\mathbf{y}}_z\right) = \sum_{i \in \{x,y,z\}} L_{\text{MSE}}\left(\mathbf{y}^i, \hat{\mathbf{y}}_i\right) + \alpha \sum_{i \in \{x,y,z\}} L_{\text{TOPO}}\left(\mathbf{y}^i, \hat{\mathbf{y}}_i\right) \tag{3}$$

These two losses have proven themselves, in previous experiments, very efficient in comparison to MSE when it comes to enforcing connectivity of image segmentation, which is our goal.

6 Conclusion

We've managed to segment the data we were provided in a way that helps the SV lab for there neurons detection as could see, now Vaa3D detects these much more accurately. Also, we started to train new networks and improve the old one to leverage the new data.

During the upcoming days, we'll work more on the project, explore new networks and losses, and leverage the amount of data we have available now. The idea of using a ResNet to extract features is brilliant and will probably allow us to increase the efficiency of our initial model, in combination with losses that enforce connectivity, we could achieve great results. Training takes a lot of time (approximately 4 days for a run), and we'll try to get the best results before the presentation.

This semester project has been a very enriching and rewarding experience. It gave me the opportunity to discover a very interesting field, which is that of biomedical image segmentation, a field that is becoming increasingly important. I have learned a lot from this project and from my supervisor which I would like to thank, he supported me along the way and we'll try to do our best before the end of the semester.

References

[1] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[2] Sihong Chen, Kai Ma, and Yefeng Zheng. Med3d: Transfer learning for 3d medical image analysis. *arXiv preprint arXiv:1904.00625*, 2019.

[3] Zaiwang Gu, Jun Cheng, Huazhu Fu, Kang Zhou, Huaying Hao, Yitian Zhao, Tianyang Zhang, Shenghua Gao, and Jiang Liu. Ce-net: Context encoder network for 2d medical image segmentation. *IEEE transactions on medical imaging*, 38(10):2281–2292, 2019.

[4] Debesh Jha, Pia H Smedsrud, Michael A Riegler, Dag Johansen, Thomas De Lange, Pål Halvorsen, and Håvard D Johansen. Resunet++: An advanced architecture for medical image segmentation. In *2019 IEEE International Symposium on Multimedia (ISM)*, pages 225–2255. IEEE, 2019.

[5] Zhengxin Zhang, Qingjie Liu, and Yunhong Wang. Road extraction by deep residual u-net. *IEEE Geoscience and Remote Sensing Letters*, 15(5):749–753, 2018.

[6] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[7] Doruk Oner, Leonardo Citraro, Mateusz Koziński, and Pascal Fua. Adjusting the ground truth annotations for connectivity-based learning to delineate. *arXiv preprint arXiv:2112.02781*, 2021.

[8] Matthias Butenuth and Christian Heipke. Network snakes: graph-based object delineation with active contour models. *Machine Vision and Applications*, 23(1):91–109, 2012.

[9] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International journal of computer vision*, 1(4):321–331, 1988.

[10] Doruk Oner, Mateusz Koziński, Leonardo Citraro, Nathan C Dadap, Alexandra G Konings, and Pascal Fua. Promoting connectivity of network-like structures by enforcing region separation. *arXiv preprint arXiv:2009.07011*, 2020.