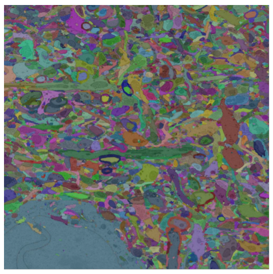
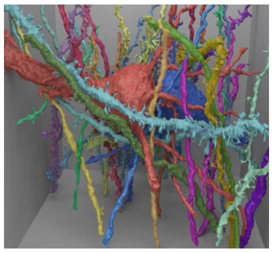
Robust Agglomeration of Labeled Neurons using 3D Skeletonization

Anonymous CVPR submission

Paper ID 446





Abstract

Advanced in electron microscopy (EM) image acquisitions has created massive petabyte datasets that are too large for complete manual segmentation. Previous automatic segmentation algorithms work by applying various techniques at the per-voxel level. Oftentimes an agglomeration strategy will build on the per-voxel results to create larger clusters of supervoxels. These results provide a good baseline but are not scalable with the size of the data. We propose a novel framework for building on top of these agglomeration strategies that scales with larger datasets using graph-based optimization strategies.

1. Introduction

Connectomics, the field of large-scale nanometer reconstruction of the wiring diagram of the brain, has created datasets petabytes in size (FIX). These datasets are too large for manual reconstruction.

- Connectomics background
- Problem Statement
- Summary of previous work (lite)
- Summary of our method
- Summary of contributions

The lowest-level frameworks in connectomics focus on per-pixel decisions using localized information. In addition, (write names) et al. use the multi-cut segmentation algorithm on the raw image data to produce segmentations. Recently, significant research focuses on developing per-pixel affinity and boundary membrane predictions using 3D convolutional neural networks (CITE VD2D3D, U-NET). Although these low-level algorithms have seen dramatic improvements in recent years, they do not fully leverage all available information. Therefore, researchers began constructing algorithms that build another level of abstraction above the per-pixel level that trained random forest classifiers using shape features at the superpixel level (CITE GALA, NeuroProof).

Although these most recent advances produce impressive improvements on the original per-pixel algorithms, they do not fully integrate the full information provided by the datasets. In this paper we provide a new framework which builds a new level of abstraction on top of the existing research. We construct a graphical representation from the outputs of the per-superpixel agglomeration strategies to further improve the merging results. We focus on the shapes of the segments to make merge and split decisions. We provide three main contributions: using a skeleton representation of the 3D segments to predict merge locations, training a 3D convolutional neural network on the segment shapes to predict merging, and a new level of abstraction on the data that considers the data in a graphical form.

2. Related Work

Much works has gone into segmentation algorithms at the per-pixel level in connectomics and more generally computer vision. Early advances in connectomics worked on generating segmentations by applying multi-cut algorithms to raw pixel values. More recently, 3D convolutional neural networks predict voxel membrane probabilities and affinities between voxels (CITE VD2D3D, U-Net, SriniNET). Recently, U-Net has been very successful in a wide range of segmentation tasks in the large medical imaging community. When generating affinities, every voxel has three outputs corresponding to the probability that the voxel (x, y, z) belongs to the same neuron as (x + 1, y, z), (x, y + 1, z) and (x, y, z + 1). Most connectomics segmentation pipelines begin by predicting membrane probabilities or voxel affinities and agglomerating voxels using this information. Zwatershed is a common 3D agglomeration strategy that builds on the output of voxel affinities (CITE ZWATERSHED). However, these methods fall short of the desired accuracy and thus the pipelines often rely on random forest classifiers to merge the oversegmented supervoxels. GALA and NeuroProof train random forest classifiers with hand designed shape features at various RESOLUTIONS. Recently, (CITE Srini) has attempted to generate the segmentation with a front-to-back artificial neural network. These flood filling networks have shown some process are expensive to run and have not been applied to large datasets. There has been extensive research into the skeletonization of 3D binary volumes in computer graphics, mathematics, and biomedical visualization. (CITE) focus on generation of topologically consistent skeletons using linear recursive methods. In the graphics community, extensive research has considered the medial axis problem of 3D renderings (CITE). In the field of connectomics, the Tree-structure Extraction Algorithm for Accurate and Robust Skeletons (TEASER) computes skeletons for individual neurons in EM images.

- Connectomics in general
- Connectomics Skeletonization (+general)
- Connectomics Segmentation
- Connectomics Agglomeration
- Connectomics Proofreading

3. Method

Our method works on top of existing EM agglomeration algorithms such as *NeuroProof* or *GALA* (CITE BOTH). First we use a 3D U-net to generate voxel affinities predicting the probability that two voxels belong to the same neuron. The *zwatershed* algorithms takes these affinities and generates an oversegmentation of the neurons (CITE). *NeuroProof* agglomerates these supervoxels by training a multi-level random forest classifier. We use a low threshold for neuroproof to generate an oversegmentation.

In order to use graph-based optimization strategies we need to generate nodes N and edges E for the graph G.

3.1. Graph Creation

3.1.1 Node Generation

The simplest method of generating nodes is to create one node for every unique segment label in a volume. However, there are many small segments that remain after NeuroProof that only introduce noise when moving to a higher level of abstraction. For now, we decide to remove these segments

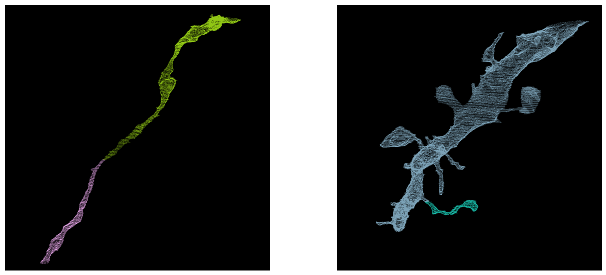


Figure 1: Two erroneously split segments that should merge together. Most segments that we want to merge have the same general structure.

from consideration and focus on the only the large segments that have expansive spans through the volume. Every segment with fewer than $20,000$ voxels is pruned from the set of considered nodes. Figure 2 shows two typical segments that each have one corresponding node in the graph G.

3.1.2 Edge Generation

The naïve edge generation approach is just to simply consider all nodes whose segmentations have a neighboring voxel. Current agglomeration approaches such as NeuroProof or GALA use this criteria for deciding which segments could potentially form one neuron. However, this produces too many edges in our graph, many of which are easily prunable. In fact, many of the segments that we need to merge have very similar structures. Consider Figure 1 which shows two erroneously split segments. In both instances the segments around the break follow the same general shape. The segment is more-or-less tubular in the close vicinity with an abrupt break at the end of segment where the next segment begins. The following algorithm identifies these locations without introducing as much "noise" as using the naïve strategy.

We begin by generating skeletons for every segment. The intuition behind this decision is that a skeleton is a simplified representation of the overarching shape of a segment. Figure 2 shows two example segments and their corresponding skeletons in white. These skeletons were generated using the Tree-structure Extraction Algorithm for Accurate and Robust Skeletons (TEASER) [6] which has been used in connectomics research previously [7]. The skeleton consists of a series of *joints* with edges between neighboring joints. Joints that have only one neighbor are also called *endpoints*. Joints that are within 50 voxels of each other are pruned to reduce unnecessary branching.

After skeletonization, we proceed to identify locations between segments that we should consider for merging. The algorithm proceeds in two passes. In the first pass we iterate over every endpoint, e, in all of the skeletons and create a set S_e where the set includes every segment that has a sin-

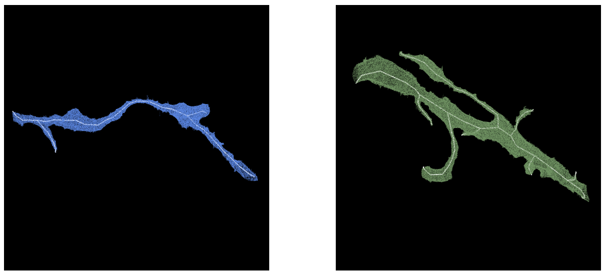


Figure 2: Two example outputs of the TEASER skeletonization algorithm.

gle voxel with t_{low}nm from the endpoint. This first pass often includes too many negative merge candidates leaving a very unbalanced set of merge/split candidates (FIX THIS). Therefore, in our second pass we iterate over all of the endpoints and their corresponding sets. We find the closest endpoint for a given segment $s \in S_e$ to the endpoint e. If the closest endpoint is less than t_{high}nm from the original endpoint e, we add an edge in the graph between these two segments. We also store the midpoint between the closest endpoint and the original endpoint e, which will be useful in the following section. Algorithm 1 provides pseudocode for this edge generation algorithm. The results in this paper follow from $t_{low} = 240nm$ and $t_{high} = 600nm$.

Algorithm 1 Edge generation function

```
function GenerateEdges($skeletons$)
    for $skeleton$ in $skeletons$ do
        $candidates$ = set()
        for $endpoint$ in $skeleton$ do
        end for
    end for
end function
```

3.2. Edge Probabilities

The previous section outlines how to generate edges between nodes in our graph structure. Here we will introduce a neural network architecture for generating probabilities that two nodes sharing an edge belong to the same neuron. The neural network takes as input only data from the input segmentation.

3.2.1 Network Architecture

The above skeletonization algorithm produces 3D locations that require further consideration for merging. To determine which of these segments should actually merge, we train a 3-D convolutional neural network. We extract a cubic region with length 1200nm centered at these locations. These cubes will provide the local information for the neural network to predict which neighboring segments belong to the same neuron. A cubic length of 1200nm provides enough local shape context for the network without introducing an excessive amount of noise. Figure 3 shows cubes of lengths 800nm, 1200nm, and 1600nm.

We transform the extracted cubes for input into the neural network. The network takes three input channels for each voxel in the 3-D volume corresponding to the following mapping. Consider a pair of segments with labels l_1 and l_2 for every voxel v in the cube of interest. The first channel is 0.5 if $v = l_1$ and -0.5 otherwise, the second channel is 0.5 if $v = l_2$ and -0.5 otherwise, and the third channel is 0.5 if $v = l_1$ or $v = l_2$ and -0.5 otherwise.

Each 3-channel volume is subsequently downsampled into an array of size $(3, 22, 68, 68)$ using nearest-neighbor interpolation. Our neural network trains on these 4-D arrays to generate probabilities that l_1 and l_2 should merge for every input.

Our network architecture has three layers of double convolutions followed by a max pooling step [1]. As described above, the input with into the network is $(22, 68, 68)$ with 3 channels. The filter size of the layers are 16, 32, and 64 with size doubling deeper into the network. The first two max pooling layers are anisotropic with pooling only in x and y to match the anisotropic nature of our EM datasets. All convolutions have kernel sizes of $(3, 3, 3)$. The output size of the final max pooling layer is $(5, 5, 5)$ with 64 channels. The output is flattened into a 1-D vector with 8000 entries which enters two fully connected layers with 512 and 1 dimensions in the output respectively. All activation functions are LeakyReLU with $\alpha = 0.001$ except for the final activation which uses a sigmoid function [2, 4]. There is a dropout of 0.2 after every pooling layer and the first dense layer. There is a dropout of 0.5 after the final dense layer. The above method uses stochastic gradient descent with Nesterov's Accelerated Gradient [5]. This optimizer has an initial learning rate of 0.01, momentum of 0.9, and a decay rate of $5 * 10^{-8}$. Figure 4 provides an overview of the proposed architecture.

3.2.2 Data Augmentation

Manual labeling of connectomics datasets is time intensive and expensive (CITE?). As we consider higher levels of abstractions of the data, the amount of useful labeled data decreases. This follows from the simple fact that manual annotation of the data happens at the per-pixel level, so starting with larger agglomerated segments leads to fewer unique human labels (THIS DOESN'T MAKE SENSE YET).

Therefore, we apply some data augmentation to the generated examples to increase the size of the training datasets. We consider all rotations of 90 degrees along the xy-plane in addition to mirrors along the x and z axes. This produces

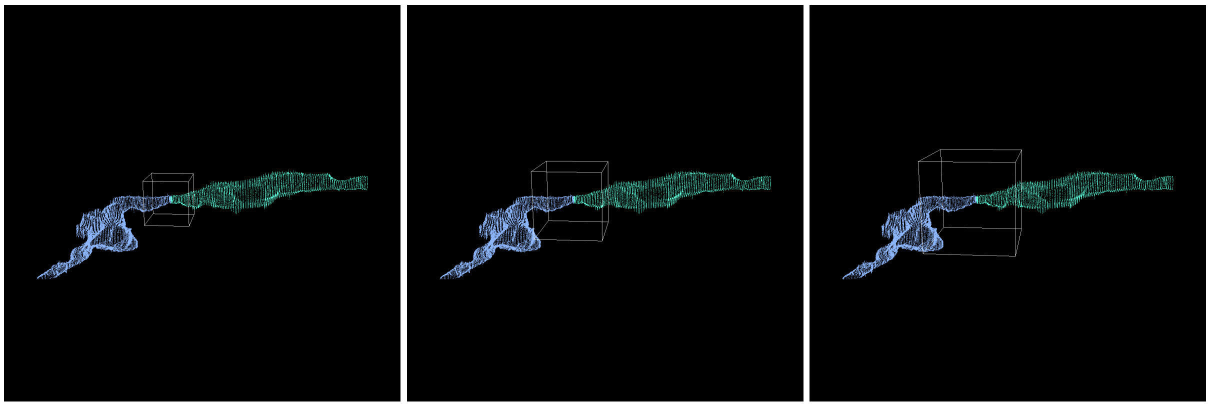


Figure 3: The white outlines show three different possible cubic sizes to input into the neural network. The left example (800nm) provides less local context than the middle example (1200nm). The right example (1600nm) extracts too large of a local region that produces noise as one of the segments leaves the bounding box only to reenter.

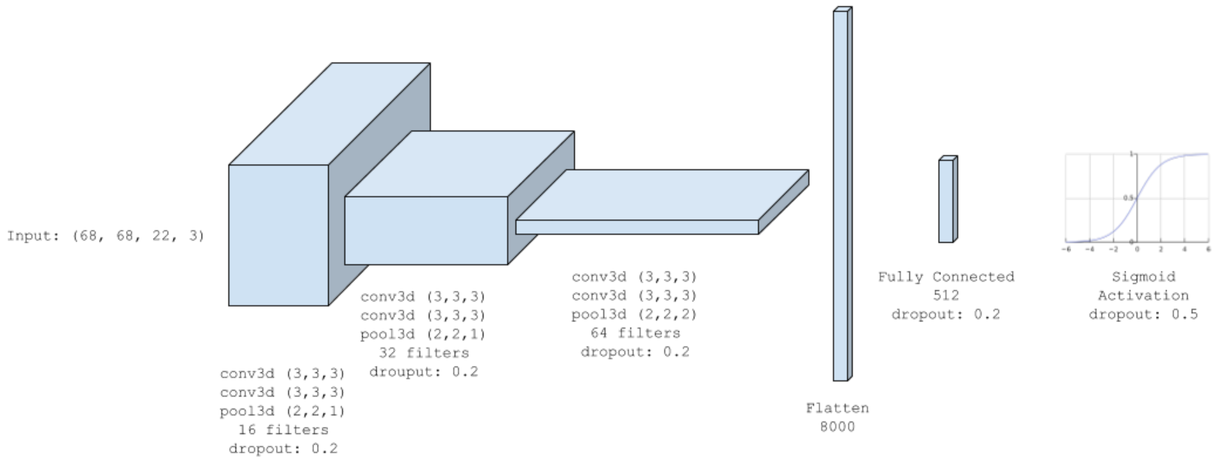


Figure 4: The architecture for the neural networks follows the *VGG* style of double convolutions followed by a max pooling operation. The number of filters doubles each layer leading to a fully connected layer and a sigmoid activation function.

an additional 16 times more training data.

3.3. Agglomeration

After constructing the above graph structure we can apply a graph-based segmentation strategy. There are many formulations of graph-based optimization strategies that provide different guarantees on their output. Neurons in the brain should be acyclic, i.e. the output shape should have a genus of zero. Current connectomics agglomeration techniques do not leverage this additional information but rather consider neighboring regions in successive order without regard to created loops. In our graph formulation we can enforce this topological property by applying a multicut partition onto the graph which generates a forest on the nodes. There are several heuristics that solve the multicut problem. For our purposes we use the Kernighan-Lin algorithm [3].

4. Evaluation

4.1. Datasets

We evaluate our methods on EM datasets. The first dataset is the Kashtui dataset which is a mouse brain (ADD IN STUFF) with an anisotropic resolution of 3nm $\times$ 3nm $\times$ 30nm. The second dataset is from FlyEM and is a (ADD STUFF HERE) with an anisotropic resolution of 3nm $\times$ 3nm $\times$ 30nm.

4.2. Preprocessing

What percent of edges are considered.

Figure (ADD FIGURE) shows a few segments that were not considered.

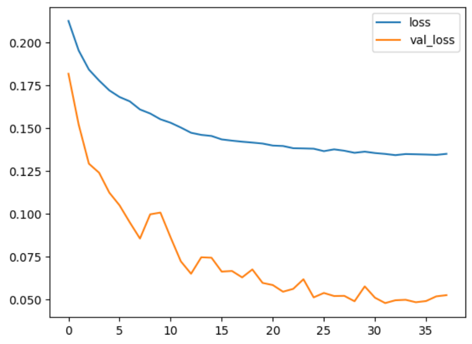


Figure 5: The training and validation loss when training the neural network on the L. Cylinder data set.

4.3. Classifier Training

Here I need to produce the precision and recall curves, (ROC) as well as the final results with thresholding. Also include the training/validation loss functions overtime

4.4. Experiments

5. Results

Here we show the results for the three main contributions for this paper: merge consideration using skeleton pruning, merge probabilities output from the trained convolutional neural network, and the final output of the pipeline.

5.1. Skeleton Pruning

5.1.1 Finding Merge Candidates

5.1.2 Finding Split Candidates

Effective pruning for merge candidates should have high precision and recall for segments that should be merged.

5.2. Neural Network Classification

5.2.1 Merge Network

Figure 5 shows the training and validation loss during the training of the neural network.

5.2.2 Split Network

5.3. Graph-based Error Correction

6. Conclusions

- Impact
- Future work

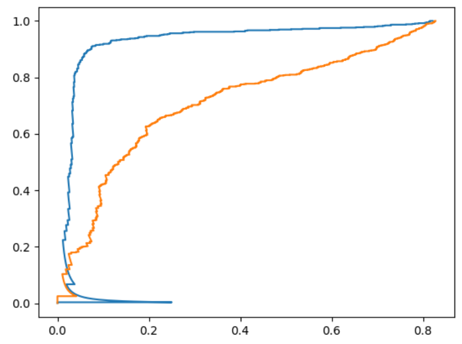


Figure 6: The ROC curve for the training and validation datasets.

References

[1] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014. 3

[2] K.-I. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192, 1989. 3

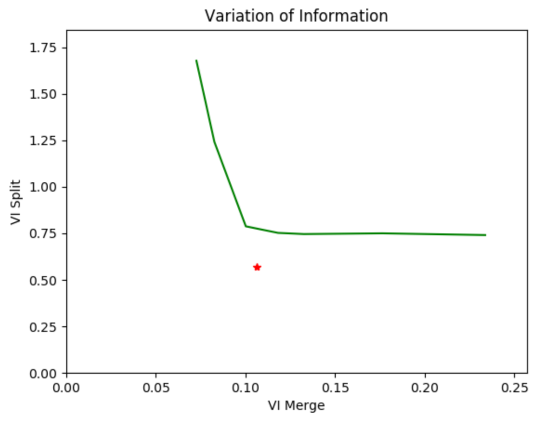
[3] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970. 4

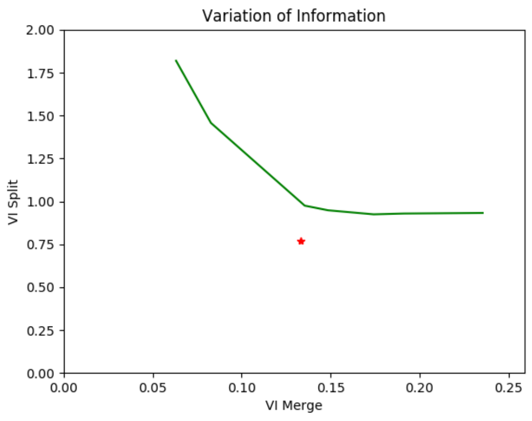
[4] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013. 3

[5] Y. Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). 3

[6] M. Sato, I. Bitter, M. A. Bender, A. E. Kaufman, and M. Nakajima. Teasar: Tree-structure extraction algorithm for accurate and robust skeletons. In *Computer Graphics and Applications, 2000. Proceedings. The Eighth Pacific Conference on*, pages 281–449. IEEE, 2000. 2

[7] T. Zhao and S. M. Plaza. Automatic neuron type identification by neurite localization in the drosophila medulla. *arXiv preprint arXiv:1409.1892*, 2014. 2


Variation of Information
VI Split
VI Merge


Variation of Information
VI Split
VI Merge