Top-Down Graph-Based Neuron Reconstruction

Anonymous CVPR submission

Paper ID 0446

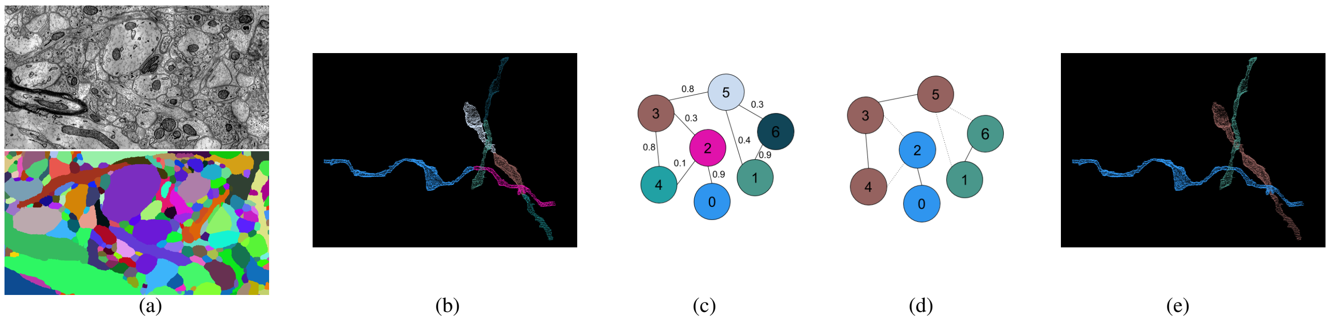

Figure 1: Outline of our approach. (a) Raw EM image data (top) and results of pixel-based segmentation and agglomeration algorithms (bottom). (b) Extracted 3D skeleton from the segmentation. (c) Simplified 3D graph with error probabilities from trained classifier (schematic). (d) Result of graph partitioning algorithm with local and global geometric features (schematic). (e) Improved 3D reconstruction.

Abstract

Advancements in electron microscopy image acquisition hour have created massive connectomics datasets in the terabyte range that make manual reconstruction of neuronal structures infeasible. Current state-of-the-art automatic methods segment neural membranes at the pixel level followed by agglomeration methods to create full neuron reconstructions. However, these approaches widely neglect global geometric properties that are inherent in the graph structure of neural wiring diagrams.

In this work, we follow bottom-up pixel-based reconstruction by a top-down graph-based method to more accurately approximate neural pathways. Using the membrane labels of the pixel-based segmentation we first generate skeletons in 3D. Using this 3D graph we train automatic classifiers for shape description to detect impossible neural pathways by looking at geometric properties. We then apply efficient graph-based optimization strategies to improve the segmentation labels. We demonstrate the performance of our approach on multiple real-world connectomics datasets with average variation of information improvement of $X\times$.

1. Introduction

[Donglai: usually, top-down means split.. but we are also bottom-up, but with different representation] The field of connectomics is concerned with reconstructing the wiring diagram of the brain at nanometer resolutions to enable new insights into the workings of the brain [16, 8]. Recent advancements in image acquisition using multi-beam serial-section electron microscopy (sSEM) have allowed researchers to produce terabytes of image data every hour [11]. It is not feasible for domain experts to manually reconstruct thousands or millions of neurons from this vast amount of image data [10]. State-of-the-art automatic reconstruction approaches use pixel-based segmentation with convolutional neural networks (CNNs) followed by agglomeration strategies [19, 24, 27, 40, 20, 29]. These *bottom-up pixel-based* algorithms produce excellent results but still fall short of acceptable error rates for large volumes.

We present a *top-down graph-based* method that builds on the outputs of bottom-up pixel-based segmentation approaches (Fig. 1a). We first extract 3D skeletons from the input segmentation (Fig. 1b) and generate a simplified 3D graph (Fig. 1c). We train a 3D CNN classifier on the agglomerated regions in the segmentation data to detect errors. During test time, we run the classifier to populate the graph edge weights with error probabilities. We then use a graph optimization algorithm to partition the graph into the final reconstruction by enforcing domain-specific global constraints from the underlying biology (Fig. 1d).

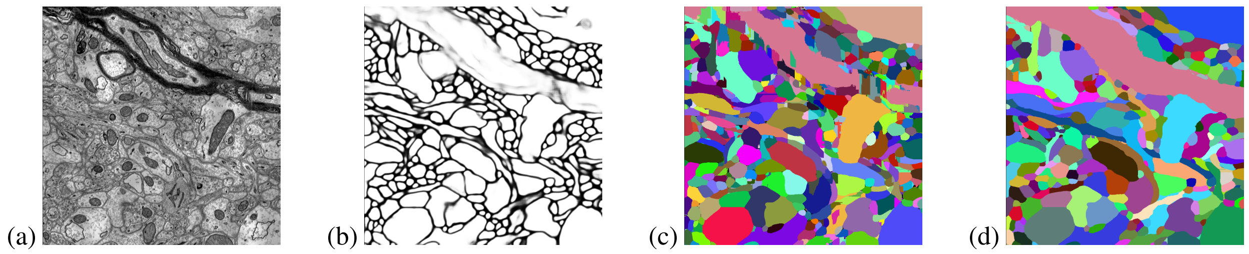

Figure 2: An overview of a pixel-based connectomics segmentation pipeline. (a) Original EM image data (b) Output of a CNN predicting voxel affinities (c) Clustering of the affinities using a watershed algorithm (d) Agglomeration of the supervoxels into larger segments.

Our approach operates at a level of abstraction above existing pixel-based methods. This allows us to leverage both local and global information to produce more accurate reconstructions. Because it uses agglomerated regions our classifier is independent of image resolution and acquisition parameters, enabling its application to isotropic and anisotropic image data without retraining. Using the 3D graph of the segmentation allows us to enforce global constraints on the reconstruction that closely follow the underlying biology, such as limiting the angles of neuron branchings to the biologically plausible range. We can also use local information such as shape priors to weight the edges in our graph. This dual approach of assessing local decisions in a global context yields accuracy improvements over the existing reconstruction methods.

This work makes the following following contributions: (1) a novel top-down graph-based method for neural reconstruction of connectomics data; (2) a region-based CNN classifier to detect errors under global constraints; (3) an empirical evaluation of our method on several connectomics datasets; (4) our method yields improved performance over the state-of-the-art on X out of Y datasets, and competitive results on the remaining data. On average, we improve the Variation of Information (VI) metric across all datasets by X percent without drastically increasing the running time.

2. Related Work

A significant amount of research in computer vision focuses on the segmentation of images [38]. Here, we review some of the most successful methods that have been applied to large-scale EM images in connectomics. Fig. 2 shows the results of a typical connectomics segmentation pipeline.

A large amount of connectomics research considers the problem of extracting segmentation information at the pixel (i.e., voxel) level from the raw EM images. Some early techniques apply computationally expensive graph partitioning algorithms with a single node per pixel [1]. However these methods do not scale to terabyte datasets. More recent methods train classifiers to predict membrane probabilities per image slice either using 2D [4, 13, 17, 19, 37] or 3D CNNs [20, 29, 36].

Oftentimes these networks produce probabilities for the affinity between two voxels (i.e., the probability that adjacent voxels belong to the same neuron). The MALIS cost function is specifically designed for generating affinities that produce good segmentations [2]. More recently, flood-filling networks produce segmentations by training an end-to-end neural network that goes from EM images directly to label volumes [14]. These flood-filling networks produce impressive accuracies but at a high computational cost.

Several pixel-based approaches generate probabilities that neighboring pixels belong to the same neuron. Often a watershed algorithm will cluster pixels into small superpixels [40]. Many methods build on top of these strategies and train random-forest classifiers to produce region-based (i.e., super-pixel) segmentations [19, 24, 26, 27, 40].

Some recent research builds on top of these region-based methods to correct errors in the segmentation [28, 41, 9]. However, to our knowledge, our method is the first to extract a 3D graph from pixel-based agglomerated segmentations for a true top-down reconstruction approach. This allows us to enforce domain-specific constraints using graph-based partitioning algorithms. Many segmentation and clustering algorithms use graph partitioning techniques [1] or normalized cuts for traditional image segmentation [15, 32, 34]. Even though graph partitioning is an NP-Hard problem [5] there are several useful multicut heuristics that provide good approximations with reasonable computational costs [12]. We use the method of Keuper et al. to partition the extracted 3D graph into the final neuron reconstruction [18].

3. Method

There are two types of errors that can occur in connectomics segmentation. The first, called a split error, occurs when there are two segments that should have been merged. The second, called a merge error, happens when one segment should be split into two. Generally, it is much more difficult to correct merge errors than to correct split errors,

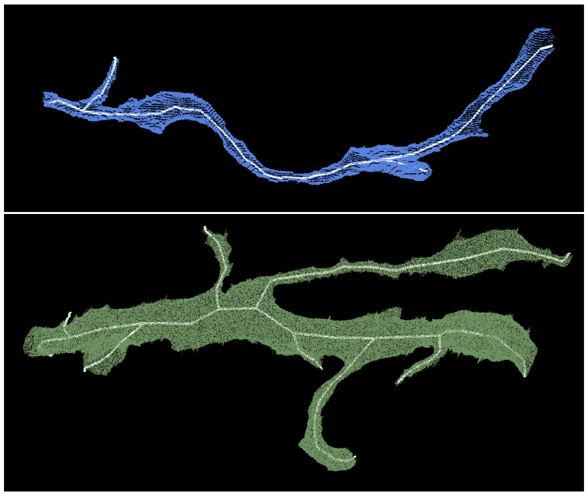

Figure 3: Example skeletons (in white) extracted from segments (blue and green) using the TEASER algorithm.

as the space of possible split proposals grow quickly [25],. Thus, most reconstruction approaches are tuned towards over-segmentation with many more split than merge errors. Our method takes as input over-segmentations of EM image volumes generated by state-of-the-art connectomics reconstruction pieplines (Sec. 4.2). Our goal is to identify locations of split errors and merge the corresponding segments automatically.

From the input segmentation we generate a graph G with nodes N and edges E with non-negative edge weights w_e. The nodes correspond to label segments from the segmentation with edges between segments considered for merging. Ideally, our graph has edges corresponding to all of the segments that were erroneously split. To compute this graph we generate a skeleton for every segment in the pixel-based segmentation. The skeleton is a simplified representation of the overall shape of the neurons. From this skeleton we identify potential merge locations and produce the corresponding edges for the graph. To find actual merges we run a classification CNN to generate edge weights that correspond to probabilites of merging. We then use a multicut heuristic to generate a partition on the graph where nodes in the same partition are assigned the same output label in the improved segmentation. We will now discuss the three major components to our framework (graph creation, edge weights assignment, and graph partitioning) in more detail.

3.1. Graph Creation

[Donglai: need some overview sentence]

3.1.1 Node Generation

The simplest node generation strategy creates one node for every unique segment label in the input volume. However, some of the millions of labels in the volume correspond to very small structures that are likely the result of segmentation errors. This typically happens in regions with noisy image data such that pixel-based methods could not generate the correct segments. It is difficult to extract useful shape features from these segments because of their small, and often random, shape. We prune these nodes from the graph by removing all segments with fewer than a threshold of t_{seg} voxels. We use $t_{seg} = 20,000$ voxels, which removed on average 56% of the segments in our datasets (Sec. 4.1). Despite the large number of segments, these regions only take up 1.6% of the total volume on average.

3.1.2 Edge Generation

A typical approach to generating edges produces an edge between all adjacent segments. Two segments l_1 and l_2 are considered adjacent if there is a pair of adjacent voxels where one has label l_1 and the other has label l_2. For example, pixel-based agglomeration methods such as NeuroProof [26] and GALA [24] consider all pairs of adjacent segments for merging. However, this method produces too many edges in the graph for graph-based optimization approaches. We identify a smaller number of pairs of segments to consider as graph edges using the following algorithm.

First, we extract a skeleton from each segment using the TEASER algorithm [30, 39]. Fig. 3 shows an example of two extracted skeletons (in white) of two segments from the label volume. These skeletons consist of a sequence of *joints*, i.e., locations that are a local maximum distance from the segment boundary, with line segments connecting successive joints. We prune the joints that are within $t_{jnt} = 50$ voxels of each other to reduce unnecessary branching. For the purposes of our algorithm, joints that have only one connected neighbor are referred to as *endpoints*. Many of the segments that are erroneously split have nearby endpoints (Fig. 4). We will make use of this fact to merge segments with the following two-pass heuristic.

In the first pass, we iterate over all endpoints e belonging to a segment S and create a set of segments $\mathbb{S}'_e$ that includes all labels that are within t_{low} voxels from e. Elements of $\mathbb{S}'_e$ are candidates for merging. However, this first pass often leads to too many candidates, requiring an additional pass for further pruning. In the second pass, we consider all of the segments in $\mathbb{S}'_e$ for every endpoint e. If a segment $S' \in \mathbb{S}'_e$ has an endpoint within t_{high} voxels of e, the segment S and S' are considered for merging. We store the midpoints between the two endpoints as the center of the potential merges in the set $\mathbb{S}_c$.

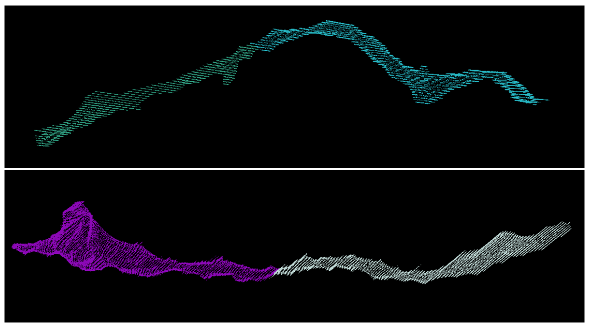

Figure 4: Two erroneously split segments. HP: split in top figure impossible to see - use brighter colors. both figures are too dark - increase brightness of segments

3.2. Edge Weights Assignment

We aim to assign non-negative edge weights w_e, where adjacent nodes (endpoints e in $\mathbb{S}_c$) connected with a big edge weight has higher chance to be merged to the same segment. Instead of using handcrafted features to compute the similarity between adjacent nodes, we train a 3D CNN classifier to learn from the manually labeled oversegmentation input volume (Sec. 4.1).

3.2.1 Classifier Input

We extract a cubic region of interest around each endpoints e in $\mathbb{S}_c$ as input to the CNN. These regions of interest provide the local information for the neural network to predict which neighboring segments belong to the same neuron. The CNN receives three input channels for every voxel in the region of interest around segments l_1 and l_2. The input in all of the channels is in the range $\{-0.5, 0.5\}$. The first channel is 0.5 only if the corresponding voxel has label l_1. The second channel is 0.5 only if the corresponding voxel has label l_2. The third channel is 0.5 if the corresponding voxel is either l_1 or l_2.

3.2.2 Network Architecture & Training

Fig. 5 provides an overview of our CNN architecture. Similar to Chatfield et al. [3] it consists of three layers of double convolutions followed by a max pooling step. The first max pooling layer is anisotropic with pooling only in the x and y dimensions. The output of this final pooling step is flattened into a 1D vector that is input into two fully connected layers. The final layer produces probabilities with a sigmoid activation function [6]. All of the other activation functions are LeakyReLU [21].

We use a stochastic gradient descent optimizer with Nesterov's accelerated gradient [23] for training. We employ dropouts of 0.2 after every pooling layer and the first dense layer, and a dropout of 0.5 after the final dense layer to prevent overfitting. We discuss all other network parameters in Sec. 4.5.

3.3. Graph Partitioning

After constructing the 3D graph we apply graph-based partitioning to compute the final segmentation. Using graph partitioning from the top down allows us to apply biological constraints on the output. Neuroscientists know that neuronal connectivity graphs in the brain are acyclic (i.e., the graphs have a genus of zero). We enforce this constraint by finding a multicut partition of the graph that generates a *forest* of nodes. A forest is a partitioning of a graph into a set of trees where no segment has a cycle. To solve this constraing multicut problem we use the method by Keuper et al. [18] that produces a feasible solution by greedy additive edge contraction. Adding additional constraints and improving the graph partition algorithm are areas for future work.

4. Evaluation

In this section, we describe the dataset, evaluation metric ... [Donglai: current organization is a bit unnatural.. too much details/overhead for the experiment setup, which can be put in the supplementary]

4.1. Datasets

We use the following two datasets for training and experimental evaluation.

Kasthuri. The Kasthuri dataset consists of scanning electron microscope images of the neocortex of a mouse brain [16]. This dataset is $5342 \times 3618 \times 338$ voxels in size. The resolution of the dataset is $3 \times 3 \times 30\,\text{nm}^3$ per voxel. We evaluate our methods using the left cylinder of this 3-cylinder dataset. We downsample the dataset in the x and y dimensions to give a final resolution of $6 \times 6 \times 30\,\text{nm}^3$ per voxel. We divide the dataset into two volumes along the x dimension where each volume is $8.0 \times 10.9 \times 10.1\,\mu\text{m}^3$.

FlyEM. The FlyEM dataset comes from the mushroom body of a 5-day old adult male Drosophila fly imaged by a focused ion-beam milling scanning electron microscopy [33]. The mushroom body in this species is the major site of associative learning. The original dataset contains a $40 \times 50 \times 120\,\mu\text{m}^3$ volume of which we use two cubes of size $10 \times 10 \times 10\,\mu\text{m}^3$. The resolution of this dataset is $10 \times 10 \times 10\,\text{nm}^3$ or 1000 voxels in each dimension in each of the two volumes.

4.2. Pixel-Based Segmentation Pipelines

The segmentation on the Kasthuri dataset was computed by agglomerating 3D supervoxels produced by the z-watershed algorithm from 3D affinity predictions [40]. A

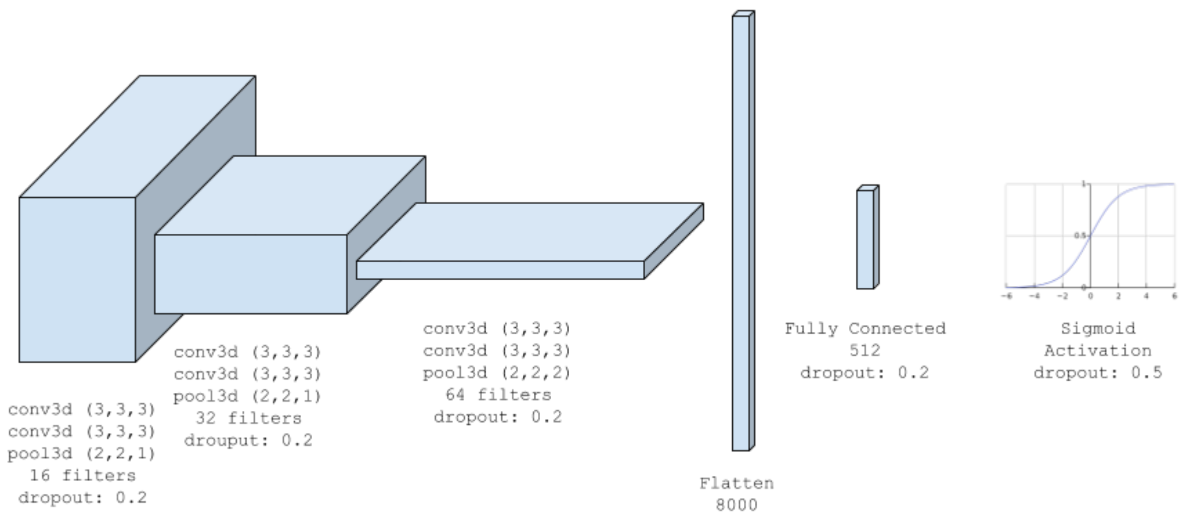

Figure 5: The architecture for our classification CNN uses double convolutions followed by max pooling. The number of filters doubles each layer, with a final fully connected layer and sigmoid activation function.

recent study by Funke et al. [31] demonstrated superior performance of such methods over existing ones on anisotropic data. We learn 3D affinities using MALIS loss with a U-net [35, 29]. We apply the z-watershed algorithm with suitable parameters to compute a 3D oversegmentation of the volume. The resulting 3D oversegmentation is then agglomerated using the technique of context-aware delayed agglomeration to generate the final segmentation [26].

For the FlyEM data we collected two $1000 \times 1000 \times 1000$ voxel ($10 \times 10 \times 10\,\mu\text{m}^3$) volumes from the authors [33]. Based on the authors' suggestion, we applied a context-aware delayed agglomeration algorithm [26] that shows improved performance on this dataset over the pipeline used in the original publication. This segmentation framework learns voxel and supervoxel classifiers with an emphasis to minimize under-segmentation error. At the same time this framework produces lower oversegmentation than standard algorithms. The algorithm first computes multi-channel 3-D predictions for membranes, cell interiors, and mitochondria, among other cell features. The membrane prediction channel is used to produce an over-segmented volume using 3D watershed, which is then agglomerated hierarchically up to a certain confidence threshold. We used exactly the same parameters as the publicly available code for this algorithm.

4.3. Error Metric

We evaluate the performance of the different methods using the split version of variance of information [22] (VI-s). Given a ground truth labeling GT and our automatically reconstructed segmentation SG. Over and under-segmentation are quantified by the conditional entropy $H(GT|SG)$ and $H(SG|GT)$, respectively. Since we are measuring the entropy between two clusterings, better VI-s scores are closer to the origin. In Fig. 6, we compute VI-s scores for the input over-segmentation result at different stopping thresholds of agglomeration.

4.4. Graph Creation Parameters

There are two essential parameters to the edge generation algorithm described in Section 3.1: t_{low} and t_{high}. Ideally, the merge candidates output by this algorithm will contains all possible positive examples with a very limited number of negative examples. After considering various thresholds, we find that $t_{low} = 240\,\text{nm}$ and $t_{high} = 600\,\text{nm}$ produce the best results considering this objective.

In our implementation we use nanometers for these thresholds and not voxels. Connectomics datasets often have lower sample resolutions in z. Using nanometers allows us to have uniform units across all of these datasets and calculate the thresholds in voxels at runtime. For example, the thresholds in voxels are $t_{low} = (40, 40, 8)$ and $t_{high} = (100, 100, 20)$ for the anisotropic Kasthuri dataset and $t_{low} = (24, 24, 24)$ and $t_{high} = (60, 60, 60)$ for the isotropic FlyEM dataset.

4.5. Classifier Training

We use the left cylinder of the Kasthuri dataset for training and validation. We train on 80% of the potential merge candidates for this volume. We validate the CNN classifier on the remaining 20% of candidates. We apply 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 mirroring along the x and z axes. This produces 16 times more training data.

We consider networks with varying input sizes, optimizers, loss functions, filter sizes, learning rates, and activation functions. The supplemental material includes information on the experiments that determined these final parameters. Table 1 provides the parameters of the final network. There are 7,294,705 learnable parameters in our final architecture. All the parameters are randomly initialized following the Xavier uniform distribution [7]. Training concluded after 34 epochs.

Parameters	Values
Loss Function	Mean Squared Error
Optimizer	SGD with Nesterov Momentum
Momentum	0.9
Initial Learning Rate	0.01
Decay Rate	$5 * 10^{-8}$
Activation	LeakyReLU ($\alpha = 0.001$)
Kernel Sizes	$3 \times 3 \times 3$
Filter Sizes	$16 \rightarrow 32 \rightarrow 64$

Table 1: Training parameters.

5. Results

5.1. Variation of Information Improvement

We compute the variation of information for our segmentations against the expert-labeled ground truth datasets. The input segmentation to our network serves as a baseline. We evaluate NeuroProof (our input) at several different thresholds to create a variation of information curve. Figure 6 shows the results on the datasets compared to the baseline (green) and an oracle (blue). The oracles sees the graph from our algorithm and correctly partitions the graph based on the ground truth. Figure 7 shows some successful merges on the Kasthuri Vol. 2. Several of these successful examples combine multiple consecutive segments. The third example shows the correction of an oversegmented dendrite. Figure 8 shows some errors made. In two of these examples the algorithm correctly predicted several merges but made just one error. In the third example, an error in the initial segmentation propagated in our pipeline. In the next three subsections we show how each step in the pipeline contributes to this final result.

5.2. Pruning via Skeletonization

Table 2 shows the results of pruning using the skeletonization heuristic. The baseline algorithm considers all adjacent regions for merging. Our method removes a significant portion of these candidates while maintaining a large number of the true merge locations. This edge pruning is essential for the graph partitioning algorithm which has a computational complexity dependence on the number of edges. Our pruning heuristic removes at least 6× the number of edges between correctly split segments on all datasets, achieving a maximum removal ratio of 20×. Equally important is the number of split errors that remain after pruning. These are the locations that we want to merge to create a more accurate reconstruction. For every dataset, the number of positive candidates remains relatively even. However, since our heuristic does not enforce an adjacency constraint of two regions when constructing edges in the graph, the difference does not indicate the number of examples excluded by pruning. In fact, our method finds a number of examples which are non-adjacent. Figure 9 shows two example segments which are split errors. The top example our algorithm missed but the segments are adjacent. The bottom example our algorithm found despite the fact that they are not adjacent.

Dataset	Baseline	After Pruning
Kasthuri Vol. 1	763 / 21242 (3.47%)	753 / 3459 (17.88%)
Kasthuri Vol. 2	1010 / 26073 (3.73%)	904 / 4327 (17.28%)
FlyEM Vol. 1	269 / 14875 (1.78%)	262 / 946 (21.69%)
FlyEM Vol. 2	270 / 16808 (1.58%)	285 / 768 (27.07%)

Table 2: The results of our pruning heuristic compared to the current baseline.

5.3. Classification Performance

Table 3 shows the precision and recall for all of the datasets. Since, our method does not rely on the image data, we can train the network on an anisotropic dataset and get impressive results on an isotropic dataset.

Dataset	Precision	Recall	Accuracy
Kasthuri Training	0.919	0.936	0.974
Kasthuri Testing	0.737	0.717	0.907
FlyEM Vol. 1	0.796	0.478	0.862
FlyEM Vol. 2	0.762	0.422	0.810

Table 3: Precision and recall for the training and three test datasets.

5.4. Graph Based Strategies

Applying a graph-based optimization strategy increases the accuracy over the CNN alone. In our test datasets, we note an increase in precision on all datasets. For our segmentation problems we prefer a higher precision since it is more difficult to correct merge-errors. Table 4 shows the changes in precision, recall, and accuracy for all three datasets in relation to the CNN.

6. Conclusions

We present a novel method for reconstructing neuronal processes in connectomics image datasets. We extend on

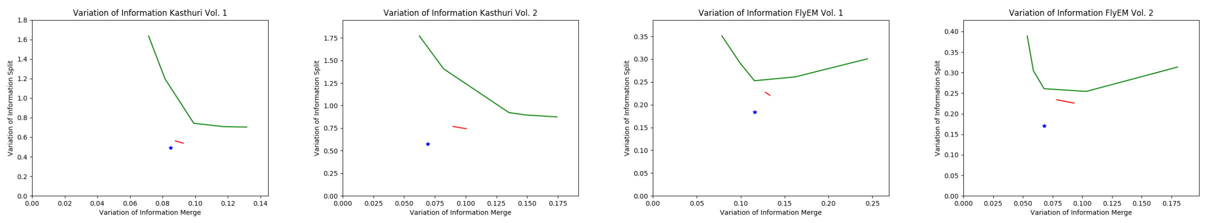

Figure 6: Our improvement (reD) on variation of information from the baseline NeuroProof segmentation (green). The blue dot represents an oracle which correctly partitions the graph.

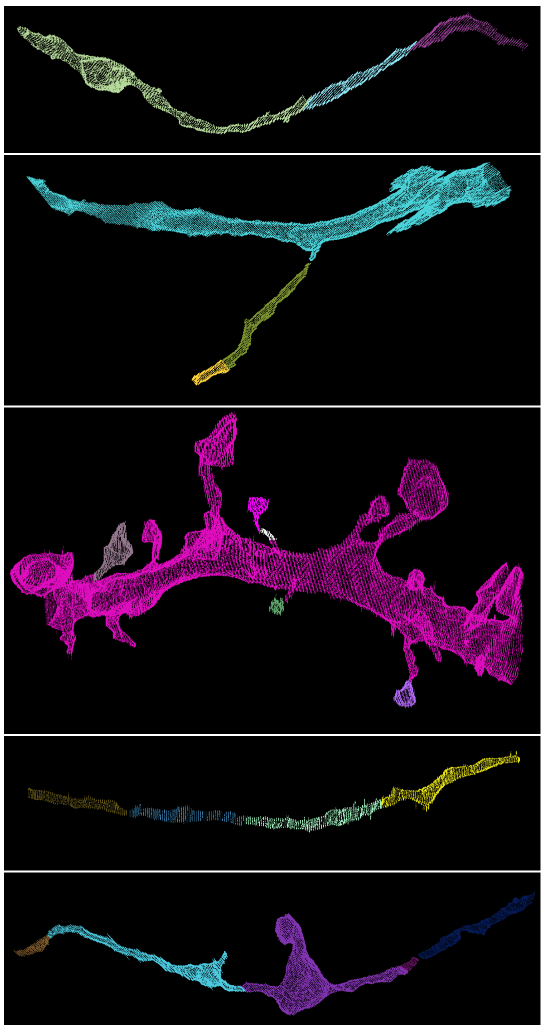

Figure 7: Some correctly predicted results from our framework.

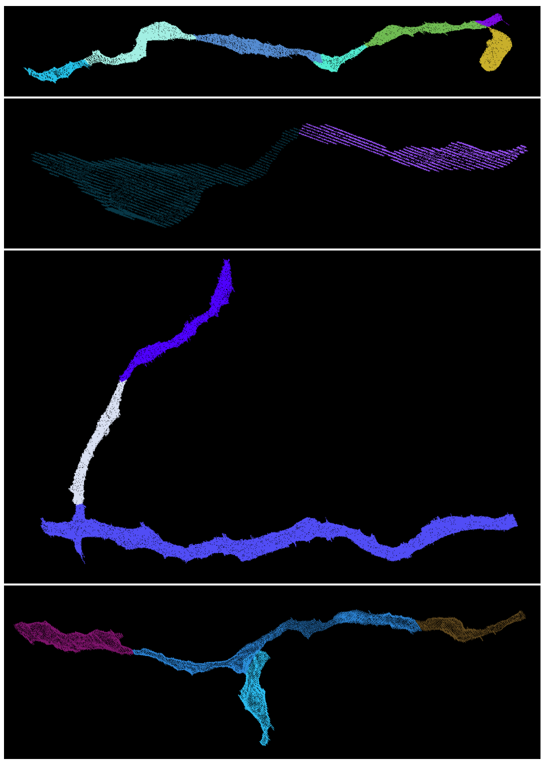

Figure 8: Mistakes made by the proposed framework.

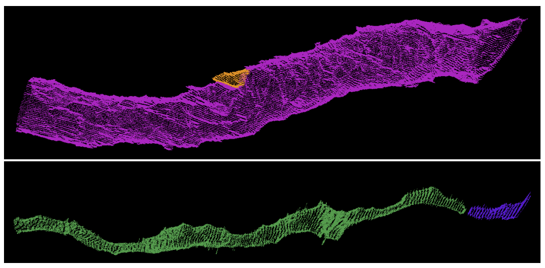

Figure 9: Example merge candidates.

top of existing region-based agglomeration strategies and show significant accuracy improvement on two datasets from two different species. By extracting skeletons from every segment in the input we can quickly produce a graph G with N nodes and E weighted edges from the input. We populate the edge weights using a 3-D CNN that, unlike previous methods, does not use the raw image data. Separating the neural network from the images allows us to train on anisotropic data and test on isotropic data with differ-

Dataset	Δ Precision	Δ Recall	Δ Accuracy
Kasthuri Training	+3.60%	-0.01%	+0.60%
Kasthuri Testing	+7.59%	-1.77%	+1.38%
FlyEM Vol. 1	+2.68%	+0.76%	+0.66%
FlyEM Vol. 2	+2.22%	-1.05%	+0.29%

Table 4: Precision and recall for the training and three test datasets.

ent sample resolution. This is exceptionally important in connectomics because of the extreme labor-intensive cost of manually segmenting ground truth data. We use existing graph partitioning strategies on the G, enabling us to enforce domain-constraints on the resulting segmentation. For us, our constraints follow from the topology of neurons. However, this framework extends to other domains with varying constraints.

References

[1] B. Andres, T. Kroeger, K. L. Briggman, W. Denk, N. Korogod, G. Knott, U. Koethe, and F. A. Hamprecht. Globally optimal closed-surface segmentation for connectomics. In *European Conference on Computer Vision*, pages 778–791. Springer, 2012. 2

[2] K. Briggman, W. Denk, S. Seung, M. N. Helmstaedter, and S. C. Turaga. Maximin affinity learning of image segmentation. In *Advances in Neural Information Processing Systems*, pages 1865–1873, 2009. 2

[3] 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. 4

[4] D. Ciresan, A. Giusti, L. M. Gambardella, and J. Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In *Advances in neural information processing systems*, pages 2843–2851, 2012. 2

[5] E. D. Demaine, D. Emanuel, A. Fiat, and N. Immorlica. Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361(2-3):172–187, 2006. 2

[6] K.-I. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192, 1989. 4

[7] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010. 6

[8] D. Haehn, J. Hoffer, B. Matejek, A. Suissa-Peleg, A. K. Al-Awami, L. Kamentsky, F. Gonda, E. Meng, W. Zhang, R. Schalek, et al. Scalable interactive visualization for connectomics. In *Informatics*, volume 4, page 29. Multidisciplinary Digital Publishing Institute, 2017. 1

[9] D. Haehn, V. Kaynig, J. Tompkin, J. W. Lichtman, and H. Pfister. Guided proofreading of automatic segmentations for connectomics. *arXiv preprint arXiv:1704.00848*, 2017. 2

[10] D. Haehn, S. Knowles-Barley, M. Roberts, J. Beyer, N. Kasthuri, J. W. Lichtman, and H. Pfister. Design and evaluation of interactive proofreading tools for connectomics. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2466–2475, 2014. 1

[11] D. G. C. Hildebrand, M. Cicconet, R. M. Torres, W. Choi, T. M. Quan, J. Moon, A. W. Wetzel, A. S. Champion, B. J. Graham, O. Randlett, et al. Whole-brain serial-section electron microscopy in larval zebrafish. *Nature*, 545(7654):345–349, 2017. 1

[12] A. Horňáková, J.-H. Lange, and B. Andres. Analysis and optimization of graph decompositions by lifted multicuts. In *International Conference on Machine Learning*, pages 1539–1548, 2017. 2

[13] V. Jain, B. Bollmann, M. Richardson, D. Berger, M. Helmstädter, K. Briggman, W. Denk, J. Bowden, J. Mendenhall, W. Abraham, K. Harris, N. Kasthuri, K. Hayworth, R. Schalek, J. Tapia, J. Lichtman, and S. Seung. Boundary learning by optimization with topological constraints. In *Proc. IEEE CVPR 2010*, pages 2488–2495, 2010. 2

[14] M. Januszewski, J. Maitin-Shepard, P. Li, J. Kornfeld, W. Denk, and V. Jain. Flood-filling networks. *arXiv preprint arXiv:1611.00421*, 2016. 2

[15] J. H. Kappes, M. Speth, G. Reinelt, and C. Schnörr. Higher-order segmentation via multicuts. *Computer Vision and Image Understanding*, 143:104–119, 2016. 2

[16] N. Kasthuri, K. J. Hayworth, D. R. Berger, R. L. Schalek, J. A. Conchello, S. Knowles-Barley, D. Lee, A. Vázquez-Reina, V. Kaynig, T. R. Jones, et al. Saturated reconstruction of a volume of neocortex. *Cell*, 162(3):648–661, 2015. 1, 4

[17] V. Kaynig, A. Vazquez-Reina, S. Knowles-Barley, M. Roberts, T. R. Jones, N. Kasthuri, E. Miller, J. Lichtman, and H. Pfister. Large-scale automatic reconstruction of neuronal processes from electron microscopy images. *Medical image analysis*, 22(1):77–88, 2015. 2

[18] M. Keuper, E. Levinkov, N. Bonneel, G. Lavoué, T. Brox, and B. Andres. Efficient decomposition of image and mesh graphs by lifted multicuts. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1751–1759, 2015. 2, 4

[19] S. Knowles-Barley, V. Kaynig, T. R. Jones, A. Wilson, J. Morgan, D. Lee, D. Berger, N. Kasthuri, J. W. Lichtman, and H. Pfister. Rhoananet pipeline: Dense automatic neural annotation. *arXiv preprint arXiv:1611.06973*, 2016. 1, 2

[20] K. Lee, A. Zlateski, V. Ashwin, and H. S. Seung. Recursive training of 2d-3d convolutional networks for neuronal boundary prediction. In *Advances in Neural Information Processing Systems*, pages 3573–3581, 2015. 1, 2

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

[22] M. Meila. Comparing clusterings by the variation of information. In *Colt*, volume 3, pages 173–187. Springer, 2003. 5

[23] Y. Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). 4

[24] J. Nunez-Iglesias, R. Kennedy, S. M. Plaza, A. Chakraborty, and W. T. Katz. Graph-based active learning of agglomeration (gala): a python library to segment 2d and 3d neuroimages. *Frontiers in neuroinformatics*, 8, 2014. 1, 2, 3

[25] T. Parag. What properties are desirable from an electron microscopy segmentation algorithm. *arXiv preprint arXiv:1503.05430*, 2015. 3

[26] T. Parag, A. Chakraborty, S. Plaza, and L. Scheffer. A context-aware delayed agglomeration framework for electron microscopy segmentation. *PLOS ONE*, 10(5):1–19, 05 2015. 2, 3, 5

[27] T. Parag, F. Tschopp, W. Grisaitis, S. C. Turaga, X. Zhang, B. Matejek, L. Kamentsky, J. W. Lichtman, and H. Pfister. Anisotropic em segmentation by 3d affinity learning and agglomeration. *arXiv preprint arXiv:1707.08935*, 2017. 1, 2

[28] D. Rolnick, Y. Meirovitch, T. Parag, H. Pfister, V. Jain, J. W. Lichtman, E. S. Boyden, and N. Shavit. Morphological error detection in 3d segmentations. *arXiv preprint arXiv:1705.10882*, 2017. 2

[29] O. Ronneberger, P. Fischer, and T. 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. 1, 2, 5

[30] 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. 3

[31] P. Schlegel, M. Costa, and G. S. Jefferis. Learning from connectomics on the fly. *Current Opinion in Insect Science*, 2017. 5

[32] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000. 2

[33] S.-y. Takemura, Y. Aso, T. Hige, A. M. Wong, Z. Lu, C. S. Xu, P. K. Rivlin, H. F. Hess, T. Zhao, T. Parag, S. Berg, G. Huang, W. T. Katz, D. J. Olbris, S. M. Plaza, L. A. Umayam, R. Aniceto, L.-A. Chang, S. Lauchie, and et al. A connectome of a learning and memory center in the adult drosophila brain. *eLife*, 6:e26975, 2017 Jul 18 2017. 4, 5

[34] S. Tatiraju and A. Mehta. Image segmentation using k-means clustering, em and normalized cuts. *Department of EECS*, 1:1–7, 2008. 2

[35] S. Turaga, K. Briggman, M. Helmstaedter, W. Denk, and S. Seung. Maximin affinity learning of image segmentation. In *Advances in Neural Information Processing Systems 22*. 2009. 5

[36] S. C. Turaga, J. F. Murray, V. Jain, F. Roth, M. Helmstaedter, K. Briggman, W. Denk, and H. S. Seung. Convolutional networks can learn to generate affinity graphs for image segmentation. *Neural computation*, 22(2):511–538, 2010. 2

[37] A. Vázquez-Reina, M. Gelbart, D. Huang, J. Lichtman, E. Miller, and H. Pfister. Segmentation fusion for connectomics. In *Proc. IEEE ICCV*, pages 177–184, Nov 2011. 2

[38] N. M. Zaitoun and M. J. Aqel. Survey on image segmentation techniques. *Procedia Computer Science*, 65:797–806, 2015. 2

[39] T. Zhao and S. M. Plaza. Automatic neuron type identification by neurite localization in the drosophila medulla. *arXiv preprint arXiv:1409.1892*, 2014. 3

[40] A. Zlateski and H. S. Seung. Image segmentation by size-dependent single linkage clustering of a watershed basin graph. *arXiv preprint arXiv:1505.00249*, 2015. 1, 2, 4

[41] J. Zung, I. Tartavull, and H. S. Seung. An error detection and correction framework for connectomics. *CoRR*, abs/1708.02599, 2017. 2