Graph-Based Neural Reconstruction from Skeletonized 3D Networks

Anonymous CVPR submission

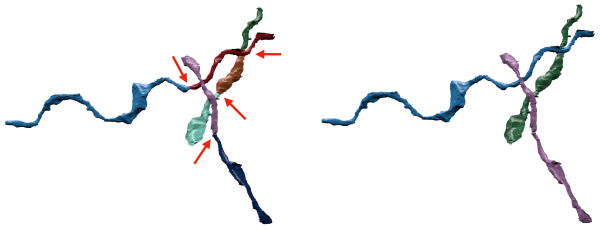
Paper ID 0446

Abstract

Advancements in electron microscopy image acquisition 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. We first generate skeletons in 3D from the neuron labels of the pixel-based segmentation. We then simplify this skeletonized 3D network into a 3D graph with nodes corresponding to labels from the segmentation and edges identifying potential locations of segmentation errors. We use a CNN classifier trained on ground truth data to generate edge weights on the 3D graph corresponding to error probabilities. We then apply a multicut algorithm to generate a partition on the graph that improves the final segmentation. Because the 3D graph is small and encodes top-down information our method is efficient and globally improves the neural reconstruction. We demonstrate the performance of our approach on multiple real-world connectomics datasets with an average split variation of information improvement of 19.3%.

1. Introduction

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 [8, 16]. 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 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, 21, 25, 28, 31, 41]. These *bottom-up pixel-based* methods produce excellent results but still fall short of acceptable error rates for large volumes.



(a) (b)

Figure 1: Example improvement of neural reconstruction. (a) We extract 3D skeletons from pixel-based segmentation algorithms to create a 3D graph representation. Edges with high segmentation error probabilities are indicated by the red arrows. (b) We improve the segmentation accuracy using a graph partitioning algorithm, leveraging both local and global information.

We present a *top-down graph-based* method that builds on the outputs of bottom-up pixel-based segmentation approaches. We first extract 3D skeleton networks from the input segmentation and generate a simplified 3D graph (Fig. 1a). We train a CNN classifier on the agglomerated regions in the segmentation data to detect errors. 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 improved reconstruction by enforcing domain-specific global constraints from biology (Fig. 1b).

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. Our method is independent of image resolution and acquisition parameters, enabling its application to isotropic and anisotropic image data without retraining. Using the 3D graph induced by the segmentation allows us to enforce global biological constraints on the reconstruc-

tion. Our dual approach of assessing local decisions in a global context yields accuracy improvements over existing reconstruction methods.

This work makes the following contributions: (1) a novel top-down method using graphs from skeletonized 3D networks for improved neural reconstruction of connectomics data; (2) a region-based CNN classifier to detect errors using the 3D graph as global constraint; (3) an empirical evaluation of our method on several connectomics datasets; (4) our method yields improved performance over a state-of-the-art pixel-based reconstruction approach on average by 19.3% without drastically increasing the running time.

2. Related Work

We review some of the most successful segmentation methods that have been applied to large-scale EM images in connectomics.

Pixel-based methods. 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, 39] or 3D CNNs [21, 31, 38].

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 networks produce impressive accuracies but at a high computational cost.

Region-based methods. Several pixel-based approaches generate probabilities that neighboring pixels belong to the same neuron. Often a watershed algorithm will then cluster pixels into super-pixels [41]. Many methods build on top of these region-based strategies and train random-forest classifiers to produce the final segmentations [19, 25, 27, 28, 41].

Error-correction methods. Some recent research builds on top of these region-based methods to correct errors in the segmentation either using human proofreading [9, 10, 20] or fully automatically [30, 42]. However, to our knowledge, our method is the first to extract a 3D graph from pixel-based segmentations for a true top-down error correction approach. This allows us to enforce domain-specific biology constraints and use efficient graph partitioning algorithms. Many segmentation and clustering algorithms use graph partitioning techniques [1] or normalized cuts for traditional image segmentation [15, 34, 36]. 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. [18] to partition the extracted 3D graph into the final neural reconstruction.

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, as the space of possible split proposals grows quickly [26]. 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 pipelines (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 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 (Fig. 2). The skeletonized 3D network is a simplified representation of the overall branching structure of the neurons. From these skeletons 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 corresponding to merge probabilities. We then use a multicut algorithm 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. 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, typically in regions with noisy raw image data. It is difficult to extract useful shape features from these segments because of their small, often random,

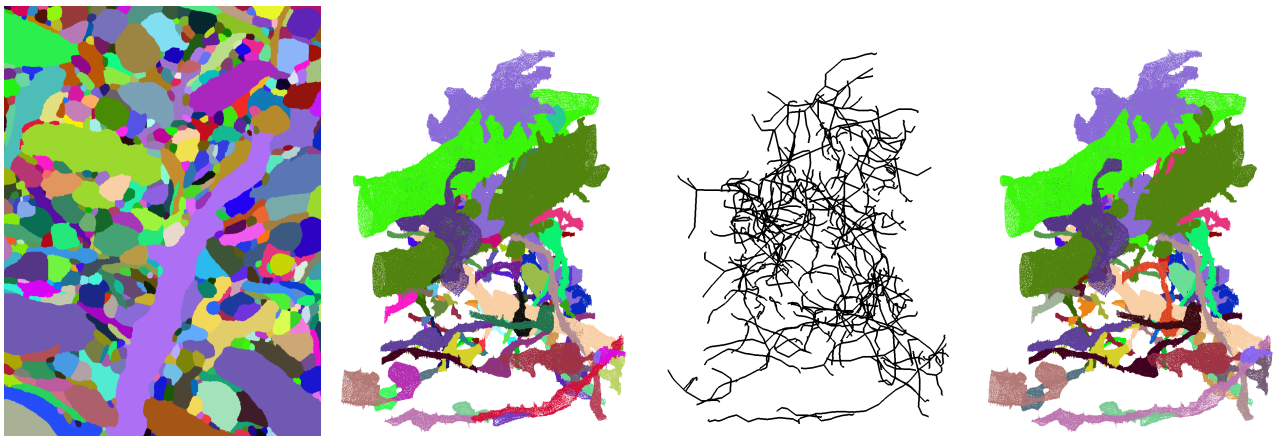


Figure 2: Outline of our approach, from left to right: result of the pixel-based segmentation and agglomeration algorithm; segments of several selected neurons from the initial segmentation; extracted skeletonized 3D network of those segments; improved 3D reconstruction of the selected segments after graph construction and partitioning with constraints.

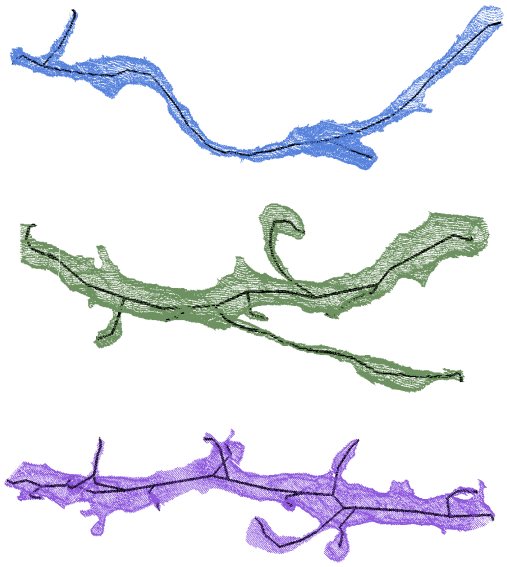


Figure 3: Example skeletons (in black) extracted from segments using the TEASER algorithm.

shape. We prune these nodes from the graph by removing all segments with fewer than a threshold $t_{seg} = 20,000$ voxels. This 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.

3.2. Edge Generation

A typical approach for generating edges produces one between all adjacent segments. Two segments l_1 and l_2 are considered adjacent if there is a pair of adjacent voxels with one labeled l_1 and the other labeled l_2. For example, pixel-based agglomeration methods such as NeuroProof [27] and GALA [25] 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 approach.

First, we extract a skeleton from each segment in the label volume using the TEASER algorithm [32, 40]. Fig. 3 shows an example of three extracted skeletons (in black). These skeletons consist of a sequence of *joints*, i.e., locations that are locally a 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. We refer to joints that have only one connected neighbor as *endpoints*. Many of the segments that are erroneously split have nearby endpoints (Fig. 4). We make use of this fact to find merge candidates with the following two-pass pruning algorithm.

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 $\mathbb{S}_c$. This algorithm produces a set of segments to consider for merging. Only these pairs have a corresponding edge in the constructed graph.

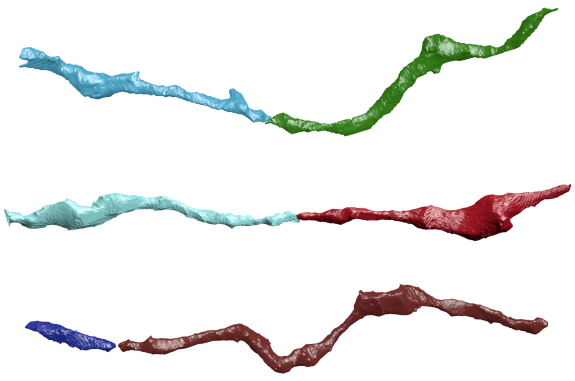


Figure 4: Three erroneously split segments.

3.3. Edge Weights Assignment

We assign edge weights w_e to each edge where the weight corresponds to the probability that two nodes belong to the same neuron. Instead of using handcrafted features to compute the similarity between adjacent nodes, we train a 3D CNN classifier to learn from the manually labeled over-segmentation input volume (Sec. 4.1). If the probability that the nodes belong to the same neuron is p_e, the edge weight $w_e = \log \frac{p_e}{1-p_e} + \log \frac{1-\beta}{\beta}$, where β is a tunable parameter that encourages over- or under-segmentation.

3.3.1 Classifier Input

We extract a cubic region of interest (ROI) around each endpoint e in $\mathbb{S}_c$ as input to the CNN. The CNN receives three input channels for every voxel in the ROI 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.

3.3.2 Network Architecture & Training

We use the CNN architecture by Chatfield et al. [3]. It consists of three layers of double convolutions followed by a max pooling step. The first two max pooling layers are 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 [22].

For training we use a stochastic gradient descent optimizer with Nesterov's accelerated gradient [24]. 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. 4.4.

3.4. Graph Partitioning

After constructing the 3D graph we apply graph partitioning using multicut to compute the final segmentation. Using top-down graph partitioning 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, i.e., 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.

4. Experimental Results

We evaluate our method by comparing it to a state-of-the-art pixel-based reconstruction approach using datasets from two different species.

4.1. Datasets

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 $3 \times 3 \times 30\,\mathrm{nm}^3$ 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 $6 \times 6 \times 30\,\mathrm{nm}^3$ per voxel. We divide the dataset into two volumes (Vol. 1 and Vol. 2) along the x dimension, where each volume is $8.0 \times 10.9 \times 10.1\,\mu\mathrm{m}^3$ or $1335 \times 1809 \times 338$ voxels.

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 [35]. The mushroom body in this species is the primary site of associative learning. The original dataset contains a $40 \times 50 \times 120\,\mu\mathrm{m}^3$ volume with a resolution of $10 \times 10 \times 10\,\mathrm{nm}^3$ per voxel. We use two cubes (Vol. 1 and Vol. 2) of size $10 \times 10 \times 10\,\mu\mathrm{m}^3$ or $1000 \times 1000 \times 1000$ voxels.

4.2. Pixel-Based Segmentations

The segmentation of the Kasthuri dataset was computed by agglomerating 3D supervoxels produced by the z-watershed algorithm from 3D affinity predictions [41]. A recent study by Funke et al. [33] demonstrated superior performance of such methods over existing ones on anisotropic data. We learn 3D affinities using MALIS loss with a U-net [31, 37]. 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 [27].

For the FlyEM data, based on the authors' suggestion [35], we applied a context-aware delayed agglomeration algorithm [27] 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 over-segmentation 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. Graph Pruning Parameters

The two parameters for the graph pruning algorithm (Sec. 3.1) are t_{low} and t_{high}. Ideally, the merge candidates output by this algorithm will contain 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 because of limitations during sample preparation. 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.

4.4. 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.

4.5. Error Metric

We evaluate the performance of the different methods using the split variation of information (VI) [23]. Given a ground truth labeling GT and our automatically reconstructed segmentation SG, over- and under-segmentation are quantified by the conditional entropies $H(GT|SG)$ and $H(SG|GT)$, respectively. Since we are measuring the entropies between two clusterings, lower VI scores are better.

4.6. Variation of Information Results

In Fig. 5, we show the VI results of the pixel-based reconstructions of the Kasthuri and FlyEM data (Sec. 4.2) for varying thresholds of agglomeration (green). We use one of these segmentations (green circle) as out input dataset with an agglomeration threshold of 0.3 for all datasets. The results from our method are shown in red for varying the β parameter. We show comparisons to an oracle (blue) that correctly partitions the graph from our method based on ground truth.

Our algorithm improves the accuracy of the reconstruction for every dataset, reducing the VI split score on average by 19.3% on the three testing datasets. Scores closer to the origin are better for this metric, and in every instance our results are below the green curve. We see significant improvements on the Kasthuri datasets (VI split reduction of 27.7% and 24.8% on the training and testing datasets respectively) and more modest improvements on the FlyEM datasets (reduction of 15.2% and 18.0%). This is because the baseline segmentation algorithm for the isotropic FlyEM data (Sec. 4.2) performs much better, reducing the potential for improvements. Isotropic datasets are easier to segment using state-of-the-art region-based methods than anisotropic ones [29].

Fig. 6 shows successful merges on the Kasthuri Vol. 2 dataset. Several of these examples combine multiple consecutive segments that span the volume. In the third example we correct the over-segmentation of a dendrite and

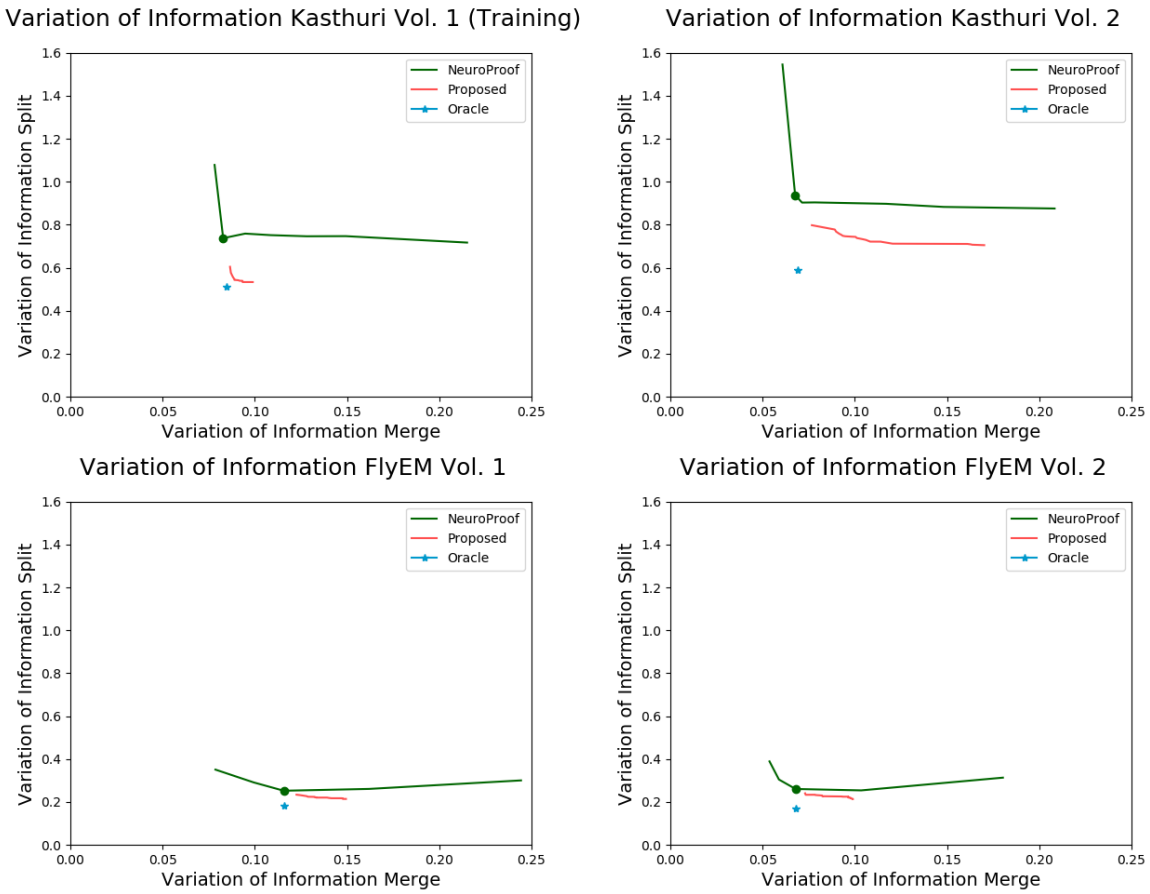


Figure 5: VI scores of our method (red) compared to the baseline segmentation (green) and an oracle (blue) that optimally partitions the graph based on ground truth. Lower scores are better. Our method improves the accuracy of the segmentation in all cases.

attached spine-necks. Fig. 7 shows typical failure cases of our method (red circles). In two of these examples the algorithm correctly predicted several merges before a single error rendered the segment as wrong. In the third example (blue circle) a merge error in the initial segmentation propagated to our output. We now analyze how each major component of our method contributes to this final result.

4.7. Graph Pruning Results

Table 2 shows the results of pruning the skeleton graph using the algorithm discussed in Sec. 3.1. This edge pruning is essential for the graph partitioning algorithm, which has a computational complexity dependence on the number of edges. 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 (e.g., 753 compared to 763). Our pruning heuristic removes at least $6\times$ the number of edges on all datasets, achieving a maximum removal rate of $20\times$.

We generate edges in our graph by using information

Dataset	Baseline	After Pruning
Kasthuri Training	763 / 21,242	753 / 3,459
Kasthuri Vol. 2	1,010 / 26,073	904 / 4,327
FlyEM Vol. 1	269 / 14,875	262 / 946
FlyEM Vol. 2	270 / 16,808	285 / 768

Table 2: The results of our graph pruning approach compared to the baseline graph with all adjacent regions. We show the number of true merge locations (e.g., 763) compared to total number of edges in the graph (e.g., 21,242) for each case.

from the skeletons. In particular, we do not enforce the constraint that edges in our graph correspond to adjacent segments. Although neurons are continuous, the EM images often have noisy spots which cause an interruption in the input segmentation. We still want to reconstruct these neurons despite the fact that the initial segmentation is non-continuous. The second and fourth examples in Fig. 6 show

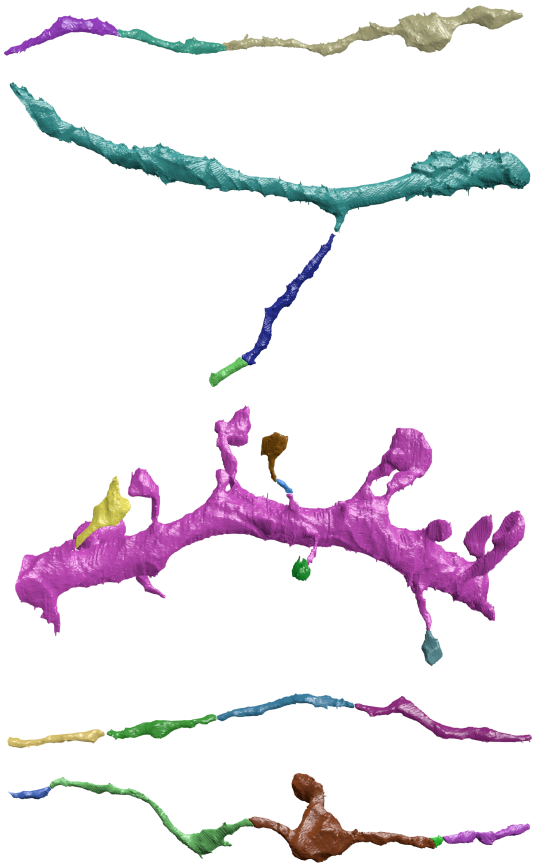


Figure 6: Segments of neurons that were correctly merged by our method.

correctly reconstructed neurons where two of the segments are non-adjacent. This is a large benefit over enforcing segment adjacency.

There are some pairs of segments which we do not consider for merging because of our reliance on the skeletons. Fig. 8 shows such a case. The endpoints of both segments are circled. In this example the small segment is carved from the larger segment in a location where there are no skeleton endpoints.

4.8. CNN Classification Results

Fig. 9 shows the receiver operating characteristic (ROC) curve of our CNN classifier for all test datasets. Since our CNN only takes as input a region of the label volume we can train on anisotropic data and test on isotropic data. This provides a major benefit given the time-intensive task of manually generating ground truth for each dataset at various resolutions.

As shown by the ROC curve, the test results on the Kasthuri data are better than the results for FlyEM. We believe this is in part because of the differences in the datasets

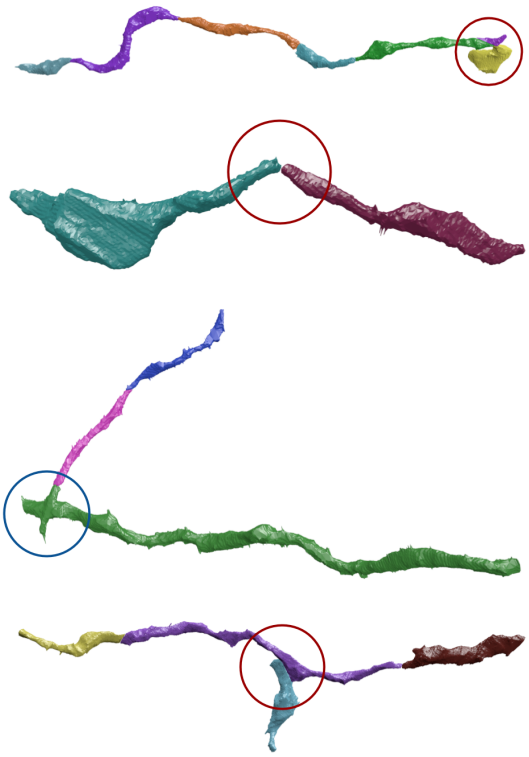


Figure 7: Circles indicate areas of wrong merges by our method (red) or by the initial pixel-based segmentation (blue).

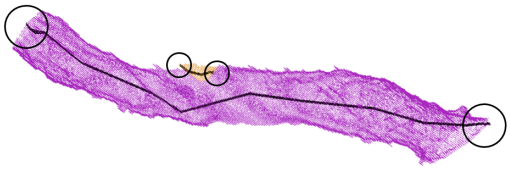


Figure 8: A false negative example of our method due to graph pruning. The distance between the endpoints (circled) of the two segments is too far to be flagged as a merge candidate.

(i.e., isotropy and xy resolution). To test this hypothesis, we also evaluate the performance of the FlyEM datasets when the network trains on FlyEM Vol. 1 and infers on FlyEM Vol. 2.[1] The blue dotted curve in the figure shows a slight performance increase in this case. However, the improvement is minor, which led us to use the CNN trained on the anisotropic data for the rest of our experiments.

[1]Since the FlyEM datasets have significantly fewer examples, we initialize the network with the weights from the Kasthuri training and have an initial learning rate of 10^{-4}.

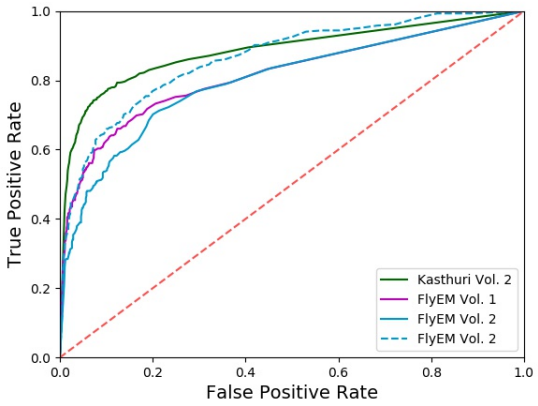


Figure 9: The receiver operating characteristic (ROC) curves of our classifier on three connectomics datasets. The classifier works best on previously unseen data of the Kasthuri volume. The dashed blue line indicates better performance on the FlyEM datasets with retraining compared to without (solid blue).

4.9. Graph Optimization Results

The graph optimization strategy using multicut increases our accuracy over using just the CNN. Table 3 shows the changes in precision, recall, and accuracy for all four datasets compared to the CNN. The precision increases on each dataset, although the recall decreases on all but one of the datasets. Since it is more difficult to correct merge errors than split errors, it is often desirable to sacrifice recall for precision. Over the three testing datasets, applying a graph-based partitioning strategy reduced the number of merge errors by 36.1%, 12.2%, and 13.6%, respectively.

Dataset	Δ Precision	Δ Recall	Δ Accuracy
Kasthuri Training	+3.61%	-0.53%	+0.60%
Kasthuri Vol. 2	+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 3: Precision, recall, and accuracy changes between CNN only and CNN paired with graph-optimized reconstructions for the training and three test datasets. The combined method results in better precision and accuracy.

5. Conclusions

We present a novel method for improved neuronal reconstruction in connectomics that extends existing pixel-based reconstruction strategies using skeletonized 3D networks. We show significant accuracy improvements on datasets from two different species. The main benefits of our approach are that it enforces domain-specific constraints at the global graph level while incorporating pixel-based classification information.

There is significant room for additional research and improvements. We can augment the graph with additional information from the image data, such as synaptic locations, cell morphology, locations of mitochondria, etc. This would allow us to enforce additional biological constraints during graph partitioning. For example, we could then enforce the constraint that a given segment only has post- or pre-synaptic connections. An augmented graph would also be helpful for splitting improperly merged segments by adding additional terms to the partitioning cost function. Finally, we believe that the benefits of top-down enhancements from graph optimization can extend beyond connectomics to other domains, such as medical image segmentation.

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. 5

[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, 2

[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] S. Knowles-Barley, M. Roberts, N. Kasthuri, D. Lee, H. Pfister, and J. W. Lichtman. Mojo 2.0: Connectome annotation tool. *Frontiers in Neuroinformatics*, (60), 2013. 2

[21] 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

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

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

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

[25] J. Nunez-Iglesias, R. Kennedy, T. Parag, J. Shi, and D. B. Chklovskii. Machine learning of hierarchical clustering to segment 2d and 3d images. *PloS one*, 8(8):e71715, 2013. 1, 2, 3

[26] T. Parag. What properties are desirable from an electron microscopy segmentation algorithm. *arXiv preprint arXiv:1503.05430*, 2015. 2

[27] 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

[28] 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

[29] S. M. Plaza, T. Parag, G. B. Huang, D. J. Olbris, M. A. Saunders, and P. K. Rivlin. Annotating synapses in large em datasets. *arXiv preprint arXiv:1409.1801*, 2014. 5

[30] 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

[31] 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, 4

[32] 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

[33] P. Schlegel, M. Costa, and G. S. Jefferis. Learning from connectomics on the fly. *Current Opinion in Insect Science*, 2017. 4

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

[35] 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

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

[37] 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. 4

[38] 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

[39] 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

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

[41] 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

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