Segmentation of Electron Microscopy Images for Connectomics

Brian Matejek
Advisor: Hanspeter Pfister
Harvard University
bmatejek@seas.harvard.edu

1. Introduction

One of the critical components of connectomics–the field concerned with reconstructing the wiring diagram of the brain at nanometer resolution–is automatic segmentation of electron microscopy (EM) images of the brain. With recent advances in EM acquisition techniques, neuroscientists can now generate a terabyte of EM image data every hour [37]. These images are typically anisotropic with 4nm resolution in the xy plane with $30 - 40$nm between slices. At this resolution, we can see all of the axon terminals and dendritic spines, and the synaptic connections between them. Accurate reconstruction of every neuron paired with synapse detection will enable us create a true graphical representation of the brain and further our understanding of the underlying circuitry. These segmentation, or label, volumes assign an integer label to every voxel where voxels have the same label only if they belong to the same neuron.

The first complete reconstruction of animal brain occurred in the 1980s with an ambitious manual effort of the C. elegans worm[CITE]. This species has [300] neurons and manual labeling required [13] years of intensive work. More recent research focuses on Drosophila flies [21][CITE JANELIA], rodents [37, 40], and even humans[CITE]. With an EM throughput of one terabyte per hour, neuroscientists can image a cubic millimeter of data (one petabyte) in six months. At this scale, the manual reconstruction efforts of the 80s are simply infeasible. Thus, researchers have turned to the deep learning revolution to provide automatic solutions to the segmentation problem.

Automatic reconstruction techniques need to be fast, scalable, and accurate. Ideally, image acquisition is the bottleneck of the connectomics pipeline which requires reconstruction efforts of a terabyte per hour [14]. We can achieve this throughput with parallelization among several GPUs with fast reconstruction techniques [11, 32]. One of the issues with these automatic techniques is that they use only local context in decision making and are agnostic about the underlying biological system. Thus, these methods make errors at scale and currently require human proofreading and other bulky error correction techniques [15, 48].

We propose a novel region merging framework that takes as input and oversegmentation of an EM volume. Our method imposes both local and global biological constrains onto the output segmentation to more close match the underlying structure of neuronal processes. Additionally, our method is independent of image resolution and acquisition parameters, enabling its application to isotropic and anisotropic data without retraining. Images generated by electron microscopes often differ in appearance because of variations in staining techniques [6]. By removing the dependence of our algorithm on the input images, we greatly reduce the need for additional costly ground truth data for each new stack of images.

2. Related Work

A significant amount of connectomics research considers the problem of extracting segmentation information at the voxel level of EM images. First, intermediate representations like boundary, affinity or binary segmentation maps are generated from the voxels. Random forests with hand-designed features [23], or 2D and 3D convolutional networks produce boundary probabilities [24, 36, 4, 8, 18, 43]. Often, the affinity between each voxel and its six neighbors are used [26, 32, 27, 7, 41]. The 3D U-Net architecture has become popular [7] and the MALIS cost function is specifically designed to re-weight affinity predictions by their contribution to the segmentation error [5]. More recently, flood-filling networks [20] produce binary segmentations from raw pixels with a recurrent convolutional network at a high computational cost. Orthogonal to the representation, model averaging [45] and data augmentation [27] methods can further improve the performance, where Lee et al. [27] surpass the estimated human accuracy on the SNEMI3D dataset.

Clustering techniques transform these intermediate representations into segmentations. Some early methods apply computationally expensive graph partitioning algorithms with a single node per superpixel [2]. Several pixel-based approaches generate probabilities that neighboring pixels belong to the same neuron. Often a watershed algorithm

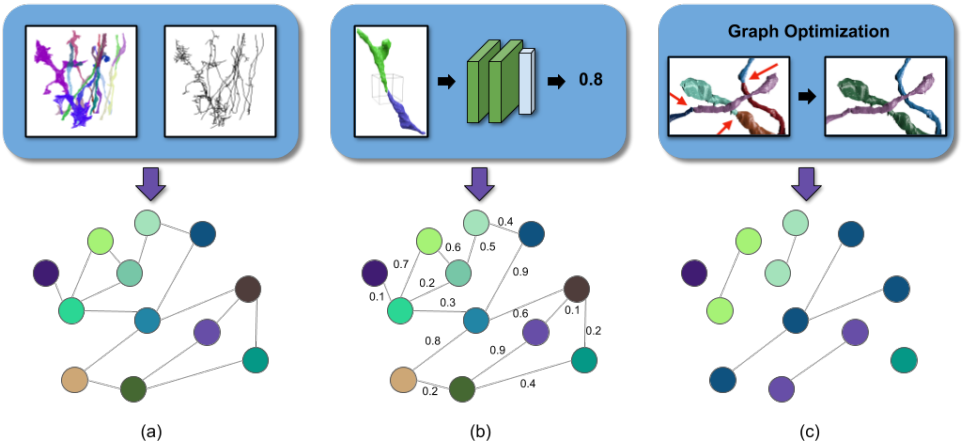


Figure 1: Our framework uses both geometric and topological biological constraints for region merging. We (a) generate a simplified graph from the skeleton representation of the input segmentation, (b) train a classifier to learn the edge weight based on the shape of the segment connection, and (c) perform graph partitioning with acyclic constraints.

will then cluster these pixels into super-pixels [47].

Region merging methods can be categorized by the similarity metric between adjacent segments and the merging strategy. For the similarity metric, Lee et al. and Funke et al. rely solely on the accuracy of the predicted affinities and define the metric as the mean affinity between segments [11, 27]. Classification-based methods generate the probability to merge two segments from handcrafted [19, 24, 29, 31, 32, 47] and learned features [4]. For the merging strategy, most methods use variants of hierarchical agglomeration [24, 29, 31, 32, 47] to greedily merge a pair of regions at a time. Jain et al. formulates the agglomeration problem as a reinforcement learning problem [19] and Pape et al. present a scalable multicut algorithm to partition superpixels with global optimization [3].

Additional research builds on top of these region-based methods to correct errors in the segmentation. This can be done either using human proofreading [15, 16, 25] or automatically [35, 48]. In both cases, available methods are pixel-based and do not include global biological constraints into the decision making process. Our method can be used as input to any existing error correction framework.

3. Preliminary Methods

Our method is illustrated in Fig. 1. From the input segmentation we generate a graph G with nodes N and edges E with weights w_e. The nodes correspond to labeled segments from the input data with edges between merge candidates. We propose the following three steps to formulate and then partition the graph while obeying constraints from the underlying biology. First, we only consider merging segments based on a skeletonized representation of the input segmentation (Fig. 1a). This enables us to reduce the number of nodes in the graph based on prior knowledge on the shape of neuronal processes. Second, we train a convolutional neural network that learns biological constraints based on the shapes of the input segmentation (Fig. 1b). This network generates probabilities that segments belong to the same neuron based only on the segmentations. An example of one such learned constraint is that neurons have small turning radii (Fig. 3). Third, we partition the graph using a lifted multicut formulation with additional acyclic constraints to enforce global biological constraints (Fig. 1c). The lifted multicut solution is globally consistent which we then augment to produce a tree-structured graph (i.e., one with no cycles).

3.1. Skeleton-Based Graph Generation

Most region merging methods create a region graph by removing small-sized segments and linking each pair of adjacent segments, which can still lead to a large graph size due to the irregular shape of neural structures. We use a skeleton representation of the segmentation to reduce the graph size with the geometric constraints on the connectivity of two adjacent segments to prune edges.

Our key observation is that if two segments belong to the same neuronal process, their skeleton end points should satisfy certain geometric constraints. To extract the skeleton from each segment, we use a variant [**?**] of the TEASER algorithm [**?**]. This skeletonization algorithm repeatedly uses Dijkstra's algorithm to find the farthest voxel from a seed location. Since this algorithm is non-linear in the number

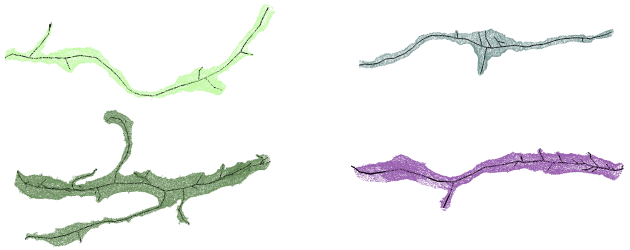


Figure 2: Example skeletons (in black) extracted from segments using a variant of the TEASER algorithm [**?**]. These skeletons not only capture the shape of the segmentation, but also provide endpoints useful for region merging proposals.

of voxels, we downsample the datasets so that there are voxel samples every $30\,\mathrm{nm}$ in each dimension. Empirically, this reduced the running time for skeletonization by $30\times$ with minimal reduction in skeleton accuracy ($\sim 5\%$ fewer branches). Fig. 2 shows four examples of extracted skeletons (in black). These skeletons consist of a sequence of *joints*, locations that are locally a maximum distance from the segment boundary, with line segments connecting successive joints. We refer to joints that have only one connected neighbor as *endpoints*. We find that approximately 70% of the segments that are erroneously split have nearby endpoints (Fig. 4).

Two segments, s_1 and s_2, receive a corresponding edge if the two following conditions hold. First, endpoints in either s_1 or s_2 are within t_{low} nm of any voxel in the other segment. Second, there are endpoints in s_1 and in s_2 that are within t_{high} nm of each other. We store the midpoints between the two endpoints as the center of the potential merge 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. We provide an empirical analysis of these parameters on the structure of the graph in the supplemental materials.

3.2. Learning-based Edge Weight

We assign an edge weight w_e to each edge corresponding to the probability that two nodes belong to the same neuronal process. We train a 3D CNN model to learn these geometric constraints for valid connection between two segments from labeled volume.

Edge Probability To predict the probabilities that two segments belong to the same neuron, we train a feed-forward convolutional network with three *VGG-style* convolution blocks [**?**] and two fully connected layers before the final sigmoid activation.

For the input of the network, 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 set $\{-0.5, 0.5\}$. 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. We do not use the raw EM image information to avoid the need to retrain the network on datasets that have been stained differently or imaged at different resolution. This reduces the need for generating costly manually-labeled ground truth.

Figure 3: Geometric constraints for region merging. We show two region merging proposals. On the left, the segments do not belong to the same neuron, as evidenced by the sharp turning radius indicated by the arrow; while on the right, the segments should be merged due to the continuity of the 3D shape. Instead of using handcrafted geometric features, we train a convolutional neural networks to automatically learn them from the ground truth labels.

Figure 4: Three erroneously split segments.

3.3. Optimization-Based Graph Partition

After graph construction, we segment the graph in a globally consistent manner while enforcing topological constraints on the output.

Lifted Multicut After constructing the graph we seek to partition it into labels where every label corresponds to a neuronal process. We formulate this graph partitioning problem as a multicut problem. There are two primary benefits to using a multicut formulation. First, the number of segments in the final graph is not predetermined but depends on the input. Second, this minimization produces globally consistent solutions (i.e., a boundary remains only if the two corresponding nodes belong to unique segments) [**?**].

We apply the algorithms of Keuper et al. [**?**] to produce a feasible solution to the multicut problem using greedy additive edge contraction. Following their example, we employ the generalized lifted multicut formulation. Traditional multicut solutions only consider the probabilities that two

adjacent nodes belong to the same segment. In the lifted extension to the problem, we can penalize non-adjacent nodes that belong to different segments. These penalties between non-adjacent nodes are called lifted edges. Ideally these lifted weights represent the probability that two nodes belong to the same neuron. However, determining such probabilities is computationally expensive. We approximate these probabilities by finding the maximal probable path between any two nodes using Dijkstra's algorithm [**?**]. This is an underestimate of the probability that two nodes belong to the same neuron since it does not consider all possible paths. Since our graphs are sufficiently small, we can generate lifted edges between all pairs of nodes.

Edge Weight We need to convert the probabilities into the following weighting scheme to solve the multicut problem with this heuristic [**?**, **?**]. Given the probability that the nodes belong to the same neuron is p_e, the edge weight w_e is defined as

$$w_e = \log \frac{p_e}{1 - p_e} + \log \frac{1 - \beta}{\beta}, \tag{1}$$

where β is a tunable parameter that encourages over- or under-segmentation. Since, there are many more lifted edges than adjacent edges, we scale down the lifted weights proportionally to their total number [3].

Topological Constraints By reformulating the segmentation problem as a graph partitioning one, we can enforce some global constraints on our result based on the underlying biology. Traditional hierarchical clustering algorithms do not rely on such constraints but consider local decisions independently. We enforce a global constraint that neurons are tree-structured and should not contain cycles. The multicut problems returns a series of "collapsed" edges between nodes that belong to the same neuron. We iterate over these edges in order of the probability of merge generated by our CNN. We "collapse" an edge only if it does not create a cycle in the graph.

4. Experiments

We evaluate our method by comparing it to a state-of-the-art pixel-based reconstruction approach using datasets from mouse and fly brains.

4.1. Datasets

Our proposed method is designed for very large connectomics datasets. Popular challenge datasets such as CREMI and SNEMI3D are simply too small for any noticeable change. The following datasets contain 4 times more volume than the CREMI datasets.

Kasthuri The Kasthuri dataset consists of scanning electron microscope images of the neocortex of a mouse brain [22]. 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 [**?**]. 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 $999 \times 999 \times 999$ voxels.

4.2. Method Configuration

Initial Segmentation The segmentation of the Kasthuri dataset is computed by agglomerating 3D supervoxels produced by the z-watershed algorithm from 3D affinity predictions [47]. A recent study by Funke et al. [11] demonstrates superior performance of such methods over existing ones on anisotropic data. We learn 3D affinities using MALIS loss with a U-Net [36, **?**]. We apply the z-watershed algorithm with suitable parameters to compute a 3D over-segmentation of the volume. The resulting 3D over-segmentation is then agglomerated using the technique of context-aware delayed agglomeration to generate the final segmentation [31]. For the FlyEM data, based on the authors' suggestion [**?**], we apply a context-aware delayed agglomeration algorithm [31] 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. 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.

Graph Generation The two parameters for the graph pruning algorithm (Sec. 3.1) are t_{low} and t_{high}. Ideally, our graph will have an edge for every over-segmented pair of labels with few edges between correctly segmented pairs. After considering various thresholds, we find that

$t_{low} = 210\,\text{nm}$ and $t_{high} = 300\,\text{nm}$ produce expressive graphs with a scalable number of nodes and edges. We explore these thresholds in greater detail in the supplemental material. During implementation, we use nanometers instead of voxels for these thresholds to have uniform units across all datasets.

Edge Weight Learning We use half 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 the candidates. Since our input does not require the image data, we can train on the anisotropic Kasthuri data and test on the isotropic FlyEM 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. We provide all parameters of the final network in the supplemental material. There are $11,456,241$ learnable parameters in our final architecture. All the parameters are randomly initialized following the Xavier uniform distribution [**?**]. Training concluded after 330 epochs.

Since our input is an existing segmentation of the EM images, there are very few training examples compared to per-pixel classifiers that train on a unique window for each voxel. The Kasthuri dataset represents a region of brain over $800\,\mu\text{m}^3$ in volume and only yields 640 positive merge examples and 4821 negative ones. To avoid overfitting our deep networks, we apply the following augmentations on the training examples. During a single batch, we randomly select ten positive and ten negative examples. With probability 0.5, an example is reflected across the xy-plane. We then rotate each example by a random angle between 0 and 360 degrees using nearest neighbor interpolation. The supplemental material contains experiments demonstrating the benefits of this augmentation strategy. We have 20,000 such examples per epoch.

Graph Partition Figure 5 shows our results for variables $\beta \in [0.5, 0.99]$. For our multicut analysis we use $\beta = 0.95$ (Sec. 3.3).

4.3. Error Metrics

We evaluate the performance of the different methods using the split variation of information (VI) [**?**]. 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 entropy between two clusterings, lower VI scores are better. The sum of these conditional entropies gives the total variation of information.

We use precision and recall to evaluate the convolutional neural network and multicut outputs. Since our method only corrects *split errors*, we define a true positive as a pair of segments that are correctly merged together after our pipeline.

5. Results

In Fig. 5, we show the VI results of the pixel-based reconstructions of the Kasthuri and FlyEM data (Sec. 4.1) for varying thresholds of agglomeration (green). We use one of these segmentations (green circle) as our 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.

Scores closer to the origin are better for this metric, and in every instance our results are below the green curve. We see improvements on each data with a reduction in total VI score of 10.4% on the Kasthuri data and 8.9% and 5.4% on the FlyEM datasets.

Fig. 6 (left) shows successful merges on the Kasthuri dataset. Several of these examples combine multiple consecutive segments that span the volume. In the third example on the left we correct the over-segmentation of a dendrite and attached spine-necks. Fig. 6 (right) shows typical failure cases of our method (red circles). In two of these examples the algorithm correctly predicts several merges before a single error renders the segment as wrong. In the third example (blue circle) a merge error in the initial segmentation propagates to our output. We now analyze how each major component of our method contributes to this final result.

5.1. Empirical Ablation Studies

Graph Generation. Table 1 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 non-linear 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., 764 compared to 974). Our pruning heuristic removes at least $3.5\times$ the number of edges on all datasets, achieving a maximum removal rate of $4.15\times$. However, there are some adjacent over-segmented labels which are not considered.

We generate edges in our graph by using information 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

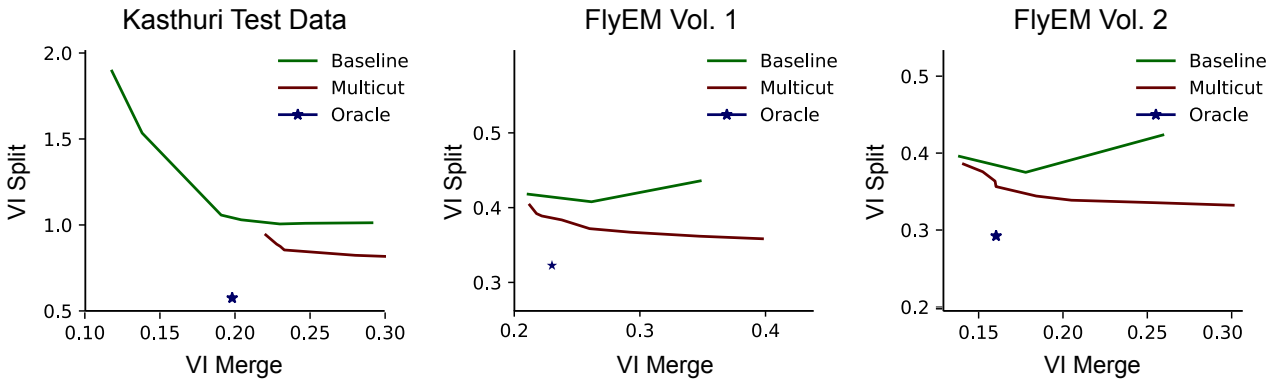


Figure 5: Segmentation benchmark results on three volumes. We compare our method (red) to the baseline segmentation (green) and an oracle (blue) that optimally partitions our constructed graph from our method. Lower VI scores are better. Our method improves the segmentation accuracy over the baseline in all cases. Note that our model is only trained on the Kasthuri training volume and it generalizes well to the FlyEM dataset.

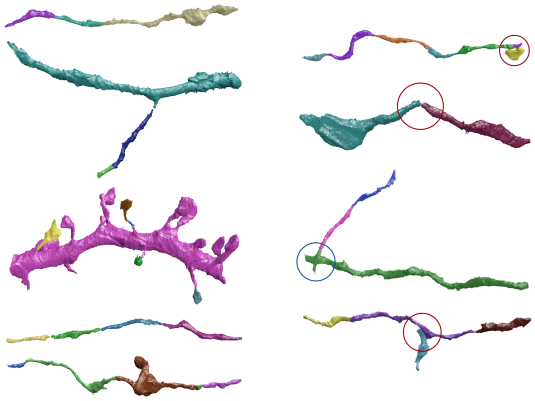


Figure 6: (left) Segments of neurons before they were correctly merged by our method. (right) Circles indicate areas of wrong merges by our method (red) or by the initial pixel-based segmentation (blue).

Table 1: 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., 974) compared to total number of edges in the graph (e.g., 25,798) for each case. The number of missed splits corresponds to the number of split errors that our method misses compared to an adjacency matrix.

Dataset	Segment Adjacency	Skeleton Pruning	Missed Splits	Gained Edges
Kasthuri	974 / 25,798	764 / 6,218	307	97
FlyEM Vol. 1	304 / 15,949	212 / 4,578	105	13
FlyEM Vol. 2	298 / 17,614	197 / 4,366	120	19

these neurons despite the fact that the initial segmentation is non-continuous. The second and fourth examples in Fig. 6 show correctly reconstructed neurons where two of the segments are non-adjacent.

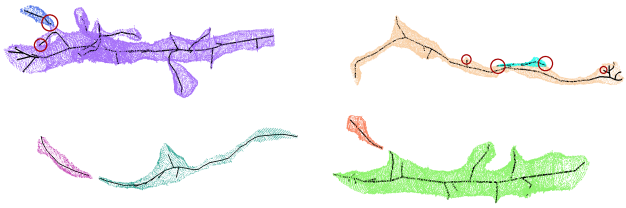


Figure 7: The top two examples correspond to segment pairs that we incorrectly prune from the graph. The distance between the circled endpoints is too great. The bottom two examples show pairs of segments that belong to the same neuron but are not adjacent in the input segmentation. However, we correctly merge these pairs.

Failure Cases There are some pairs of segments which we do not consider for merging because of our reliance on the skeletons. Fig. 7 shows two such cases with the closest endpoints circled. In the right example the small segment is carved from the larger segment in a location where there are no skeleton endpoints. There are on average 177 such examples in our datasets.

Edge Weight Learning Data augmentation at test time can improve accuracy results [27, 45]. When computing the probability to merge two segments, we randomly rotate and flip the examples three times (in the same fashion as training augmentation). The supplemental material contains experiments showing the trade-offs between increased accuracy and runtime when using these augmentation strategies.

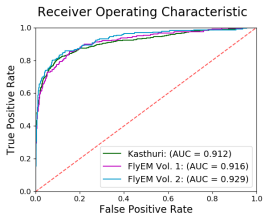


Figure 8: The receiver operating characteristic (ROC) curves of our classifier on three connectomics datasets. The classifier works best on previously unseen data of the Kasthuri volume.

Fig. 8 shows the receiver operating characteristic (ROC) curve of our CNN classifier for all test datasets. As shown by the ROC curve, the test results on the FlyEM data are better than the results for Kasthuri. In part this comes from the disparity in the number of positive to negative merge candidates in the two graphs. The network easily classifies most of the negative examples leaving only a few difficult examples to predict. Since there are more negative examples in relation to the positive examples in the FlyEM data, the ROC curve is greater.

Our neural network does not take as input the images or affinities so we can transfer network weights from one dataset to the next. The classification results are comparable on the FlyEM datasets despite the fact that the data varies in resolution and even animal type from the training set.

Graph Partition The graph optimization strategy using multicut increases accuracy over using just the CNN. Table 2 shows the changes in precision, recall, and accuracy for all four datasets compared to the CNN with both the multicut and lifted multicut formulations. These results correspond to $\beta = 0.95$ (Sec. 3.2). The precision increases on each dataset, although the recall decreases on each 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 increased the precision by 31.9%, 40.9%, and 27.8% respectively.

Table 2: 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. The lifted multicut extension provides very slight improvements in recall and accuracy over these three datasets.

	Multicut			Lifted Multicut		
Dataset	**Δ Precision**	**Δ Recall**	**Δ Accuracy**	**Δ Precision**	**Δ Recall**	**Δ Accuracy**
Kasthuri	31.94%	-36.24%	0.71%	-1.01%	0.60%	0.02%
FlyEM Vol. 1	40.87%	-42.37%	1.26%	0.35%	0.85%	0.04%
FlyEM Vol. 2	27.80%	-44.95%	0.33%	0.54%	0.92%	0.04%

5.2. Computational Performance

System All performance experiments ran on an Intel Core i7-6800K CPU 3.40 GHz with a Titan X Pascal GPU. All code is written in Python and is freely available (link omitted for review). We use the Keras deep learning library for our neural networks with Theano backend and cuDNN 7 acceleration for CUDA 8.0.

Every step in our framework is fast enough for large connectomics datasets. On our training data, after downsampling, each segment was skeletonized in 0.56 seconds on average. There are 4451 such segments resulting in a total running time of 2,492 seconds on an 800 megavoxel volume. From here, it took just over 31 seconds to generate the edges for our graph. On our hardware, inference on the neural network has a throughput of 30 examples per second. Lastly, the multicut algorithm ran for 25.28 seconds while the lifted multicut variant took 36.5 seconds on the graph from this dataset.

6. Proposed Work

There are significantly more biological constraints that we plan to use in the future. Once we improve our synaptic classifiers, we can ensure that we do not merge dendritic spines with axons. By stipulating that a segment on one side of the cell body can only have either pre- or post-synaptic connections, we will prevent undersegmentation that merges multiple neurons together. Additionally, once we have a classifier to label each neuron as inhibitory or excitatory, we can prevent the merging of two different types of neurons. Both of these additional constraints will help with the current problem associated with large-scale reconstruction. Namely, that a small percentage of merge errors results in a tangled mess of multiple neurons per segment.

To further prevent this problem, we will augment our work to correct split errors as well. We will generate locations from splits with the following algorithm. First, we will locate places where a single skeleton joint has three immediate neighbors (Fig **??**). We will choose two of the neighbors as seed locations for a watershed algorithm that will run on the affinities constrained to the segment. This will generate a potential split location between the two seeds. We will then extract a region of interest around this split and use that cube as input into our previously trained merge classifier. If the classifier says that the two segments should not merge, we will accept the split as is and divide the segment. Otherwise we will ignore this split candidate and keep the segment intact.

A benefit of this approach is that we will have one CNN to determine both merge and split decisions. This allows us to better tune our parameters with the scarce number of training examples from error correction. Figure **??** shows our proposed framework that corrects both merge and split

errors.

7. Compression

An often overlooked aspect of these segmentation datasets is how to efficiently store the label volumes. These label volumes are often 32- or 64-bit to account for the massive number of neurons that interweave through a cubic millimeter. Uncompressed, a cubic millimeter of segmentation data is nearly 20 petabytes and oftentimes there are a series of saved interim segmentations. Simply put, compression techniques are needed for fast transmission and cost-efficient storage.

General-purpose compression schemes [9, 10, 12, 28, 30, 33, 38, 42, 44, 46] are not optimized for this data. With Compresso, an algorithm published in MICCAI 2017, we exploit the typical characteristics of label volumes such as large invariant regions without natural relationship between label values. These properties render 2D image compression schemes inadequate since they rely on frequency reduction (using e.g., wavelet or discrete cosine transform) and value prediction of pixels based on local context (differential pulse-code modulation) [34, 39]. Color space optimization strategies in video codecs [1] also have no effect on label volumes, even though the spatial properties of a segmentation stack (z-axis) are similar to the temporal properties of video data (time-axis). A compression scheme designed specifically for label volumes is part of the visualization software Neuroglancer [13]. This method exploits segmentation homogeneity by creating small blocks with N labels and reducing local entropy to $\log_2 N$ per pixel. Lookup tables then decode the values $[0, N)$ to the original 64-bit labels.

Compresso works by decoupling the two important components across the image stack: per-segment shape and per-pixel label. To encode the segment shapes, we consider the boundary pixels between two segments. Removing the per-pixel labels, we produce a boundary map for each slice where a pixel (x, y, z) is 1 if either pixel at $(x + 1, y, z)$ or $(x, y + 1, z)$ belongs to a different segment (Fig 9, left). The boundary map is divided into non-overlapping congruent 3D windows. If there are n pixels per window, each window w is assigned an integer $V_w \in [0, 2^n)$ where V_w is defined as:

$$V_w = \sum_{i=0}^{n-1} \mathbb{I}(i) 2^i, \tag{2}$$

and $\mathbb{I}(i)$ is 1 if pixel i is on a boundary and 0 otherwise. Figure 9, left, shows an example segmentation with a window size of $4 \times 4 \times 1$.

A priori, each window could take any of 2^n distinct values, and therefore require n bits to encode without further manipulation. However, boundaries in segmentation images are not random, and many of these values never appear. Indeed, we find that a small subset of high-frequency V_w values accounts for most windows, allowing for significant compression. Figure 9, right, shows the 100 most common windows for a representative connectomics dataset. These 100 frequently occurring windows account for approximately 82% of the over 1.2 million V_w values in this dataset. Nearly all of these windows correspond to simple lines traversing through the window. For contrast, we also provide 5 randomly generated windows that never occur in the dataset.

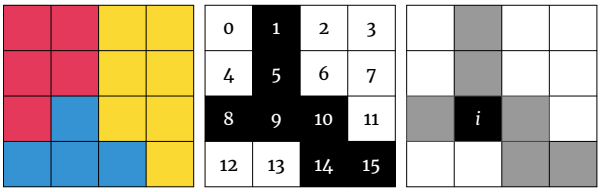
We define N as the number of distinct V_w representing all of the windows in an image stack. We construct an invertible function $f(V_w) \rightarrow [0, N)$ to transform the window values into a smaller set of integers. For all real-world segmentations $N \ll 2^n$; however, we assume no constraint on N in order to guarantee lossless compression. With this function, each V_w requires $\log_2 N$ bits of information to encode. This is fewer than the initial number of bits so long as $N \leq 2^{n-1}$.

So far we have focused exclusively on transforming the boundary map of an image segmentation. However, the per-pixel labels themselves are equally important. The boundary map divides each image slice into different segments. By design, all pixels in the same segment have the same label so we store only one label per segment for each slice. We use a connected-component labeling algorithm to store one label per segment [17]. The algorithm labels all pixels clustered within a component m from M section labels.

Thus far, we have assumed the boundaries provide enough information to reconstruct the entire segmentation. Pixels not on a segment boundary are easily relabeled. However, more care is needed for pixels on the segment boundaries. Consider Figure 9, left, which depicts a difficult boundary to decode. If a boundary pixel has a non-boundary neighbor to the left or above, then that pixel merely takes on the value of that neighbor. However, pixel i requires more care since its relevant neighbors are both boundary pixels. We simply just store the label values for indeterminate pixels like i.

To decompress the data, we first reconstruct the boundary map from the windows that we extracted. From there, we can run the same deterministic connected-components algorithm per slice. We can traverse through the saved labels to fill in all non-boundary labels. To determine the per-pixel labels for every boundary pixel, we iterate over the entire dataset in raster order. Any boundary pixel (x, y, z) with a non-boundary neighbor at $(x-1, y, z)$ or $(x, y-1, z)$ shares the same per-pixel label. If both relevant neighbors are boundaries (i.e., like pixel i in Fig. 9) we extract the value we had stored.

Our compression scheme outperforms all existing meth-



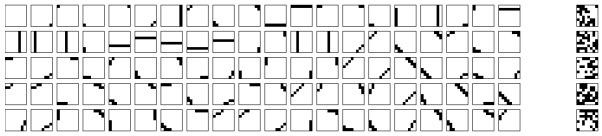


Figure 9: Compresso works by dividing up the segmentation data into small blocks. On the left we show the intersection of three segments in a $4 \times 4 \times 1$ window. We extract a boundary map from this segmentation to transform the window into an integral value between 0 and $2^{16} - 1$. The location i requires some additional bookkeeping. On the right are the 100 most common $8 \times 8 \times 1$ windows accounting for $\sim 82\%$ of the volume on a representative dataset.

ods. We follow our scheme with LZMA which uses sophisticated probabilistic bit prediction strategies and achieve ratios of $700\times$ on average. We compressed an 18 terabyte label volume of 100 microns cubed to 26.2 gigabytes, a ratio of $\sim 670\times$.

8. 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.

In the future, these methods can be adjusted to apply additional domain constraints. We can augment the graph with more information from the image data, such as synaptic connections, cell morphology, and locations of mitochondria. This would allow us to match other 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 be helpful for splitting improperly merged segments by adding additional terms to the partitioning cost function. Using skeletons we can apply biological constraints on the topology for neurons that are improperly merged. 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] L. Aimar, L. Merritt, E. Petit, et al. x264-a free h264/avc encoder, 2005. 8

[2] 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. 1

[3] T. Beier, C. Pape, N. Rahaman, T. Prange, S. Berg, D. D. Bock, A. Cardona, G. W. Knott, S. M. Plaza, L. K. Scheffer, et al. Multicut brings automated neurite segmentation closer to human performance. *Nature methods*, 14(2):101, 2017. 2, 4

[4] J. A. Bogovic, G. B. Huang, and V. Jain. Learned versus hand-designed feature representations for 3d agglomeration. *arXiv preprint arXiv:1312.6159*, 2013. 1, 2

[5] 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. 1

[6] K. L. Briggman and D. D. Bock. Volume electron microscopy for neuronal circuit reconstruction. *Current opinion in neurobiology*, 22(1):154–161, 2012. 1

[7] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger. 3d u-net: learning dense volumetric segmentation from sparse annotation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 424–432. Springer, 2016. 1

[8] 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. 1

[9] Collet and Turner. Smaller and faster data compression with zstandard, url: https://code.facebook.com/posts/1658392934479273/smaller-and-faster-data-compression-with-zstandard/, 2016. Accessed: 23-October-2016. 8

[10] P. Deutsch and J.-L. Gailly. Zlib compressed data format specification version 3.3. Technical report, 1996. 8

[11] J. Funke, F. D. Tschopp, W. Grisaitis, C. Singh, S. Saalfeld, and S. C. Turaga. A deep structured learning approach towards automating connectome reconstruction from 3d electron micrographs. *arXiv preprint arXiv:1709.02974*, 2017. 1, 2, 4

[12] Google. Brotli compression format, url: https://github.com/google/brotli, 2016. Accessed: 11-October-2016. 8

[13] Google. Neuroglancer compression, url: https://github.com/google/neuroglancer/blob/master/src/neuroglancer/sliceview/compressed_segmentation/readme.md, 2016. Accessed: 21-October-2016. 8

[14] 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

[15] 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. 1, 2

[16] 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. 2

[17] L. He, Y. Chao, K. Suzuki, and K. Wu. Fast connected-component labeling. *Pattern Recognition*, 42(9):1977–1987, 2009. 8

[18] 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. 1

[19] V. Jain, S. C. Turaga, K. Briggman, M. N. Helmstaedter, W. Denk, and H. S. Seung. Learning to agglomerate superpixel hierarchies. In *Advances in Neural Information Processing Systems*, pages 648–656, 2011. 2

[20] M. Januszewski, J. Maitin-Shepard, P. Li, J. Kornfeld, W. Denk, and V. Jain. Flood-filling networks. *arXiv preprint arXiv:1611.00421*, 2016. 1

[21] T. Jovanic, C. M. Schneider-Mizell, M. Shao, J.-B. Masson, G. Denisov, R. D. Fetter, B. D. Mensh, J. W. Truman, A. Cardona, and M. Zlatic. Competitive disinhibition mediates behavioral choice and sequences in drosophila. *Cell*, 167(3):858–870, 2016. 1

[22] N. Kasthuri, K. J. Hayworth, D. R. Berger, et al. Saturated reconstruction of a volume of neocortex. *Cell*, 162(3):648–661, 2015. 4

[23] 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. 1

[24] 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

[25] 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

[26] 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

[27] K. Lee, J. Zung, P. Li, V. Jain, and H. S. Seung. Superhuman accuracy on the snemi3d connectomics challenge. *arXiv preprint arXiv:1706.00120*, 2017. 1, 2, 6

[28] M. Lehmann. Liblzf, url: http://oldhome.schmorp.de/marc/liblzf.html, 2016. accessed: 13-October-2016. 8

[29] 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. 2

[30] M. Oberhumer. Lzo real-time data compression library. *User manual for LZO version 0.28, url: http://www.infosys.tuwien.ac.at/Staff/lux/marco/lzo.html (February 1997)*, 2005. 8

[31] 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, 4

[32] 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

[33] I. Pavlov. Lzma sdk (software development kit), 2007. 8

[34] G. Roelofs and R. Koman. *PNG: the definitive guide*. O'Reilly, Inc., 1999. 8

[35] 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

[36] 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, 4

[37] R. Schalek, D. Lee, N. Kasthuri, A. Suissa-Peleg, T. R. Jones, V. Kaynig, D. Haehn, H. Pfister, D. Cox, and J. W. Lichtman. Imaging a 1 mm3 volume of rat cortex using a multibeam sem. In *Microscopy and Microanalysis*, volume 22, pages 582–583. Cambridge Univ Press, 26 July 2016. 1

[38] J. Seward. bzip2, 1998. 8

[39] A. Skodras, C. Christopoulos, and T. Ebrahimi. The jpeg 2000 still image compression standard. *IEEE Signal processing magazine*, 18(5):36–58, 2001. 8

[40] A. Suissa-Peleg, D. Haehn, S. Knowles-Barley, V. Kaynig, T. R. Jones, A. Wilson, R. Schalek, J. W. Lichtman, and H. Pfister. Automatic neural reconstruction from petavoxel of electron microscopy data. *Microscopy and Microanalysis*, 22:536, 2016. 1

[41] 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. 1

[42] Vandevenne and Alakuijala. Zopfli compression algorithm, url: https://github.com/google/zopfli, 2016. Accessed: 11-October-2016. 8

[43] 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. 1

[44] T. A. Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, 1984. 8

[45] T. Zeng, B. Wu, and S. Ji. Deepem3d: approaching human-level performance on 3d anisotropic em image segmentation. *Bioinformatics*, 33(16):2555–2562, 2017. 1, 6

[46] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE transactions on Information Theory*, 24(5):530–536, 1978. 8

[47] 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. 2, 4

[48] J. Zung, I. Tartavull, and H. S. Seung. An error detection and correction framework for connectomics. *CoRR*, abs/1708.02599, 2017. 1, 2