Segmentation of Electron Microscopy Images for Connectomics

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

1. Introduction

The field of connectomics is concerned with the reconstruction of the wiring diagram of the mammalian brain. In order to achieve this ambitious goal, we need images at nanometer resolution to see the structure of the neurons and the synaptic connections between them. Recent advances in electron microscopy acquisition techniques enable a throughput of a terabyte of image data every hour [35]. Manual reconstruction of this image data is simply infeasible and requires automatic segmentation methods. These automatic methods label every voxel of the image with a 64-bit integer value where two voxels have the same label only if they belong to the same neuron. As the field progresses, our goal is to reconstruct a cubic millimeter of brain which will result in over 18 petabytes of segmentation data alone [38].

Three major challenges arise as the image datasets scale to a petabyte and more. First, the segmentation methods must be accurate enough to extract the proper connections between neurons. In particular, the dendritic spines [check,CITE] which contain the postsynaptic density can be only [XX] nanometers thick. Second, automatic segmentation must be fast – ideally matching the throughput of the electron microscope [14]. The current state-of-the-art automatic reconstruction pipelines are either too slow for such large datasets or too inaccurate at scale. Third, the label volumes require $8\times$ the number of bytes as the already massive image volumes. Storing such data is expensive and transferring the data is slow. To cut costs and delays, we need compression methods to reduce data sizes.

2. Related Work

Automatic Segmentation. 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 [21], or 2D and 3D convolutional networks produce boundary probabilities [22, 34, 4, 7, 18, 41]. Often, the affinity between each voxel and its six neighbors are used [24, 30, 25, 6, 39]. The 3D U-Net architecture has become popular [6] 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 [43] and data augmentation [25] methods can further improve the performance, where Lee et al. [25] 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 will then cluster these pixels into super-pixels [45].

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 [25, 11]. Classification-based methods generate the probability to merge two segments from handcrafted [22, 27, 30, 45, 29, 19] and learned features [4]. For the merging strategy, most methods use variants of hierarchical agglomeration [22, 27, 30, 45, 29] 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 [16, 15, 23] or automatically [33, 46]. 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.

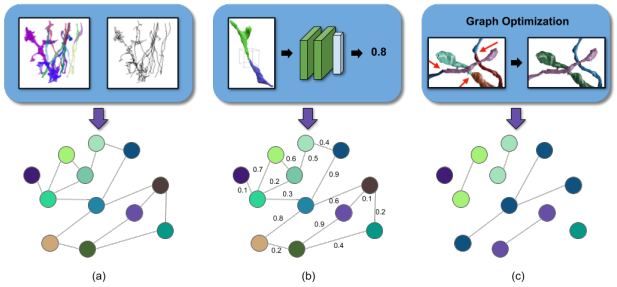

Figure 1: Our framework uses both geometric and topological biological constraints for region merging. We (a) generate a simplified graph from the skeleton representation of the input segmentation, (b) train a classifier to learn the edge weight based on the shape of the segment connection, and (c) perform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

Compression. The literature currently lacks efficient compression of label volumes. General-purpose compression schemes [9, 44, 26, 28, 42, 36, 31, 40, 12, 8] are not optimized for this data. In this paper, 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) [32, 37]. 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. We compare the Neuroglancer scheme with our method.

3. Preliminary Methods

Biologically-Constrained Error Correction 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).

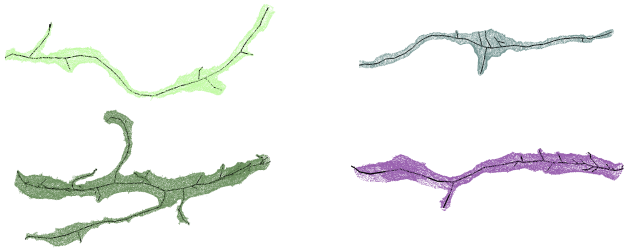

Figure 2: Example skeletons (in black) extracted from segments using a variant of the TEASER algorithm [**?**]. These skeletons not only capture the shape of the segmentation, but also provide endpoints useful for region merging proposals.

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 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

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.

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 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.

Compression Segmentation datasets contain two important pieces of information across the image stack: per-segment shape and per-pixel label. Decoupling these two components allows for better compression on each.

Boundary Encoding. 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. 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 **??** 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 **??** 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 $N \leq 2^{n-1}$. We create two arrays that store the per-segment shape encoding: `WindowValues[]` contains the value $f(V_w)$ for every window w and `ValueMapping[]` contains the reverse mapping from $[0, N) \rightarrow [0, 2^n)$ based on the function f. Long sequences of 0s in `WindowValues[]` are reduced using run-length encoding.

Per-Pixel Label Compression. 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. We store the original label for a segment m in slice z in $\texttt{Labels}_z\texttt{[m]}$. We concatenate these arrays for every image slice to create a variable `Labels[]`.

Exceptions. Thus far, we have assumed the boundaries described in Section 3.3 provide enough information to reconstruct the entire segmentation. Pixels not on a segment boundary are easily relabeled using the `Labels[]` array. However, more care is needed for pixels on the segment boundaries. Consider Figure **??**, 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, the pixel i requires more care since its relevant neighbors are both boundary pixels. If a non-boundary neighbor pixel shares a label with the undetermined pixel, we add the offset to that neighbor to an array `IndeterminateValues[]`. Otherwise we add that per-pixel label.

Metadata. We construct a data structure containing the two per-segment shape and two per-pixel label arrays. The last component of the data structure is the `Header`, which contains the dimensions of the original data, the window size, and the size of the arrays. Compresso could be improved by further compressing the individual components of the encoding (e.g., Huffman encoding the V_w values). We achieve strong overall compression by using a second-stage general compression scheme such as LZMA (Sec. **??**).

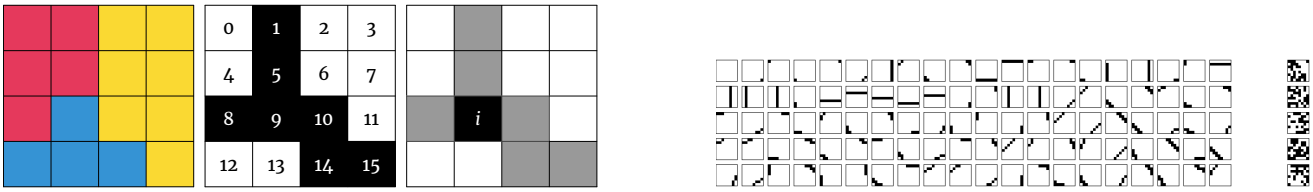

Figure 5: Compresso works by dividing up the segmentation data into small blocks. On the left we show the intersection of three segments in a $4 \times 4 \times 1$ window. We extract a boundary map from this segmentation to transform the window into an integral value between 0 and $2^{16} - 1$. The location 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.

3.4. Decoding

The first step in decoding the data is to reconstruct the boundary map. We iterate over every pixel, determine the corresponding window w, and retrieve the encoded window value $f(V_w)$ from the `WindowValues[]` array. These values range from 0 to $N - 1$ and correspond to an index in `ValueMapping[]` that contains the original V_w value. After decoding V_w, the value of pixel i in window w equals $V_w \wedge 2^i$.

After reproducing the boundary map, we execute the same deterministic connected-components algorithm per slice as when encoding. Each component in the boundary map receives a label between 0 and $M - 1$. Using the `Labels[]` array, we can easily translate these component labels into the original per-pixel labels for every slice. 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 we consider the next unused value in the `IndeterminateValues[]` array and update this pixel's label.

4. Preliminary Results

5. 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.

6. Conclusion

References

[1] L. Aimar, L. Merritt, E. Petit, et al. x264-a free h264/avc encoder, 2005. 2

[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. 1, 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

[5] K. Briggman, W. Denk, S. Seung, M. N. Helmstaedter, and S. C. Turaga. Maximin affinity learning of image segmenta-

tion. In *Advances in Neural Information Processing Systems*, pages 1865–1873, 2009. 1

[6] 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

[7] 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

[8] 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. 2

[9] P. Deutsch and J.-L. Gailly. Zlib compressed data format specification version 3.3. Technical report, 1996. 2

[10] M. Fredman and M. Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 345–354. ACM, 1989. 5

[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

[12] Google. Brotli compression format, url: https://github.com/google/brotli, 2016. Accessed: 11-October-2016. 2

[13] Google. Neuroglancer compression, url: https://github.com/google/neuroglancer/blob/master/src/neuroglancer/sliceview/compressed_segmentation/readme.md, 2016. Accessed: 21-October-2016. 2

[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

[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. 1

[17] L. He, Y. Chao, K. Suzuki, and K. Wu. Fast connected-component labeling. *Pattern Recognition*, 42(9):1977–1987, 2009. 4

[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. 1

[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] 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

[22] 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

[23] 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. 1

[24] 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

[25] 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

[26] M. Lehmann. Liblzf, url: http://oldhome.schmorp.de/marc/liblzf.html, 2016. accessed: 13-October-2016. 2

[27] 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

[28] 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. 2

[29] 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. 1

[30] 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

[31] I. Pavlov. Lzma sdk (software development kit), 2007. 2

[32] G. Roelofs and R. Koman. *PNG: the definitive guide*. O'Reilly, Inc., 1999. 2

[33] 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. 1

[34] 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

[35] 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

[36] J. Seward. bzip2, 1998. 2

[37] A. Skodras, C. Christopoulos, and T. Ebrahimi. The jpeg 2000 still image compression standard. *IEEE Signal processing magazine*, 18(5):36–58, 2001. 2

[38] 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

[39] 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

[40] Vandevenne and Alakuijala. Zopfli compression algorithm, url: https://github.com/google/zopfli, 2016. Accessed: 11-October-2016. 2

[41] 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

[42] T. A. Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, 1984. 2

[43] 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

[44] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE transactions on Information Theory*, 24(5):530–536, 1978. 2

[45] 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

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