An Error Detection and Correction Framework for Connectomics

Jonathan Zung
Princeton University
jzung@princeton.edu

Ignacio Tartavull
Princeton University
tartavull@princeton.edu

H. Sebastian Seung
Princeton University
sseung@princeton.edu

Abstract

We define and study error detection and correction tasks that are useful for 3D reconstruction of neurons from electron microscopic imagery, and for instance segmentation more generally. The error detection task takes as input the raw image and a binary mask representing a candidate object. The desired output is a map of split and merge errors in the object. The error correction task is formulated as object mask pruning. The input is the raw image and a candidate object mask. The task is to reduce the object mask to generate the true object. We train multiscale 3D convolutional networks to perform both tasks, and then show how the nets can be combined to achieve significant improvements in segmentation accuracy. Namely, the error detecting net is used to create the object mask that is the input to the error correcting net.

1 Introduction

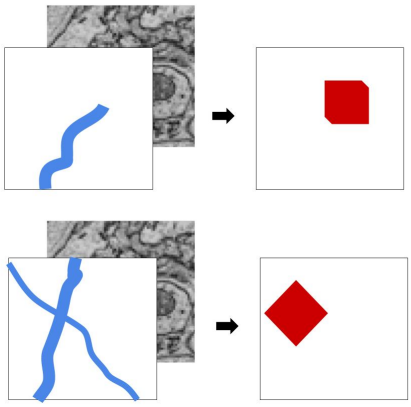
While neuronal circuits can be reconstructed from volumetric electron microscopic imagery, the process has historically [24] and even recently [22] been highly laborious. One of the most time-consuming reconstruction tasks is the tracing of the brain's "wires," or neuronal branches. This task is an example of instance segmentation, and can be automated through computer detection of the boundaries between neurons. Convolutional nets were first applied to neuronal boundary detection a decade ago [23, 5]. Since then convolutional nets have become the standard approach, and the accuracy of boundary detection has become impressively high [25, 2, 10, 3].

Given the low error rates, it becomes helpful to think of subsequent processing steps in terms of modules that detect and correct errors. In the error detection task (Figure 1a), the input is the raw image and a binary mask that represents a candidate object. The desired output is a map containing the locations of split and merge errors in the candidate object. Related work on this problem has been restricted to detection of merge errors only by either hand-designed [13] or learned [19] computations.

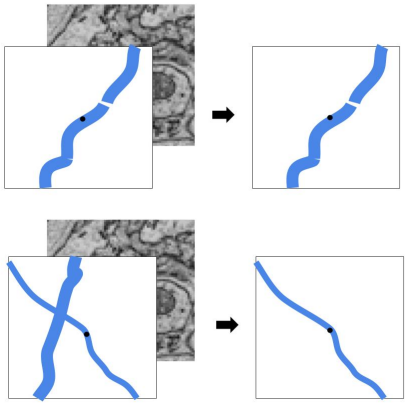
In the error correction task (Figure 1b), the input is again the raw image and a binary mask that represents a candidate object. The candidate object mask is assumed to be a superset of a true object, which is the desired output. With this assumption, error correction is formulated as *object mask pruning*. Object mask pruning can be regarded as the splitting of undersegmented objects to create true objects. In this sense, it is the opposite of agglomeration, which merges oversegmented objects to create true objects [6, 16]. Object mask pruning can also be viewed as the *subtraction* of voxels from an object to create a true object. In this sense, it is the opposite of a flood-filling net [8, 7], each iteration of which is the *addition* of voxels to an object to create a true object. Flood-filling nets are closely related to other work on instance segmentation in connectomics [13] and computer vision [20, 18]. The task of generating an object mask de novo from an image has also been studied in computer vision [17].

We implement both error detection and error correction using 3D multiscale convolutional networks. One can imagine multiple uses for these nets in a connectomics pipeline. For example, the error-

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.



(a) Input to the error detector on the left and the corresponding error map on the right. In the merge error, the error map is roughly diamond shaped since a pixel in the error map is red iff a window centred on it touches both objects. The actual input and output are 3d volumes.



(b) Desired input and output for the error correction network. The black dot marks the central pixel in each window. In the first case there is nothing to prune, while in the second case the object not overlapping the central pixel is erased.

detecting net could be used to reduce the amount of labor required for proofreading by directing human attention to locations in the image where errors are likely. This labor reduction could be substantial because the declining error rate of automated segmentation has made it more time-consuming for a human to find an error.

We find it especially interesting and natural to combine the error-detecting and error-correcting nets in the following way. To create the candidate object mask for the error-correcting net from a baseline segmentation, one can simply take the union of all erroneous segments as found by the error-detecting net. Since the error rate in the baseline segmentation is already low, this union is small and it is easy to select out a single object.

We contend that our approach decomposes the neuron segmentation problem into two strictly easier pieces. First, we hypothesize that recognizing an error is much easier than producing the correct answer. Indeed, humans are often able to detect errors using only morphological cues such as abrupt terminations of axons, but may have difficulty actually finding the correct extension.

On the other hand, if the error-detection network has high accuracy and the initial set of errors is sparse, then the error correction module only needs to prune away a small number of irrelevant supervoxels from the candidate mask described above. This contrasts with the flood-filling task which involves an unconstrained search for a few supervoxels to add. Given that most supervoxels are *not* a part of the object to be reconstructed, an upper bound on the object is more valuable than a lower bound. Furthermore, selective application of the error correction module near likely errors loosens our computational budget.

In this paper, we support the intuition above by demonstrating high accuracy detection of both split and merge errors. We also demonstrate a complete implementation of the stated error detection-correction framework, and report significant improvements upon our baseline segmentation.

2 Error detection

2.1 Task specification

Given a single segment in a proposed segmentation presented as an object mask Obj, the error detection task is to produce a binary image called the *error map*, denoted $Err_{p_x \times p_y \times p_z}(Obj)$. The definition of the error map depends on a choice of a window size $p_x \times p_y \times p_z$. A pixel i in the error map is 0 if and only if the restriction of the input mask to a window centred at i of size $p_x \times p_y \times p_z$

is pixel-wise equal to the restriction of some object in the ground truth. Observe that the error map is sensitive to both split and merge errors.

A smaller window size allows us to localize errors more precisely. On the other hand, if the window radius is less than the width of a typical boundary between objects, it is possible that two objects participating in a merge error never appear in the same window. These merge errors would not be classified as an error in any window.

We could use a less stringent measure than pixel-wise equality that disregards small perturbations of the boundaries of objects. However, our proposed segmentations are all composed of the same building blocks (supervoxels) as the ground truth segmentation, so this is not an issue for us.

We define the *combined error map* as $\sum_{Obj} Err(Obj) * Obj$ where $*$ represents pointwise multiplication. In other words, we restrict the error map for each object to the object itself, and then sum the results. The figures in this paper show the *combined error map*.

2.2 Network architecture

We take a fully supervised approach to error detection. We implement error detection using a multiscale 3D convolutional network. The architecture is detailed in Appendix B. Its design is informed by experience with convolutional networks for neuronal boundary detection (see [10]) and reflects recent trends in neural network design [21, 4]. Its field of view is $P_x \times P_y \times P_z = 318 \times 318 \times 33$ (which is roughly cubic in physical size given the anisotropic resolution of our dataset). The network computes (a downsampling of) $Err_{46\times46\times7}$. At test time, we perform inference in overlapping windows and conservatively blend the output from overlapping windows using a maximum operation.

We trained two variants, one of which takes as input only Obj, and another which additionally receives as input the raw image.

3 Error correction

3.1 Task specification

Given an image patch of size $P_x \times P_y \times P_z$ and a candidate object mask of the same dimensions, the *object mask pruning* task is to erase all voxels which do not belong to the object overlapping the central pixel.

At test time, the input binary mask contains the union of all objects with a detected error.

3.2 Network architecture

Yet again, we implement error correction using a multiscale 3D convolutional network. The architecture is detailed in Appendix B. One difficulty with training a neural network to reconstruct the object containing the central pixel is that the desired output can change drastically as the central pixel moves between objects. We use an intermediate representation whose role is to soften this dependence on the location of the central pixel. The desired intermediate representation is a k dimensional vector $v(x, y, z)$ at each point (x, y, z) such that points within the same object have similar vectors and points in different objects have different vectors. We transform this vector field into a binary image M representing the object overlapping the central pixel as follows:

$$M(x, y, z) = \exp\left(-||v(x, y, z) - v(P_x/2, P_y/2, P_z/2)||^2\right).$$

When an over-segmentation is available, we replace $v(P_x/2, P_y/2, P_z/2)$ with the average of v over the supervoxel containing the central pixel. This trick makes it unnecessary to centre our windows far away from a boundary, as was necessary in [8]. Note that we backpropagate through the transform M, so the vector representation may be seen as an implementation detail and the final output of the network is just a (soft) binary image.

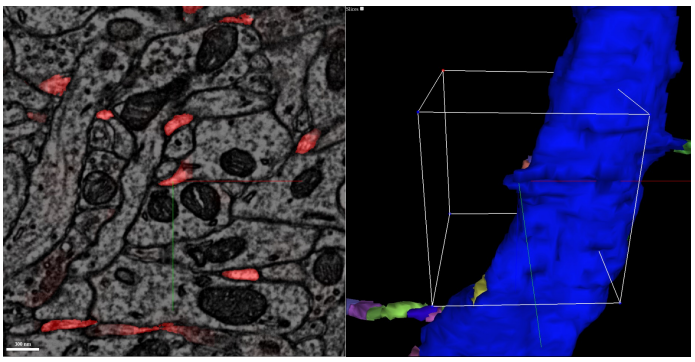


Figure 2: An example of a mistake in the initial segmentation. The dendrite is missing a spine. The red overlay on the left shows the combined error map (defined in Section 2.1); the stump in the centre of the image was clearly marked as an error.

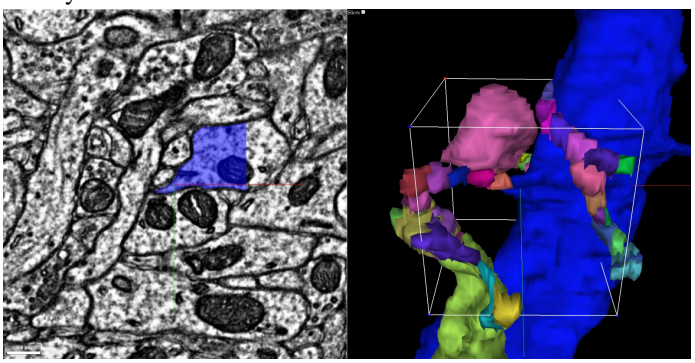


Figure 3: The right shows all objects which contained a detected error in the vicinity. This is the binary mask which is provided as input to the error correction network. For clarity, these objects were clipped to lie within the white box representing the field of view of our error correction network. The output of the error correction network is overlaid in blue on the left.

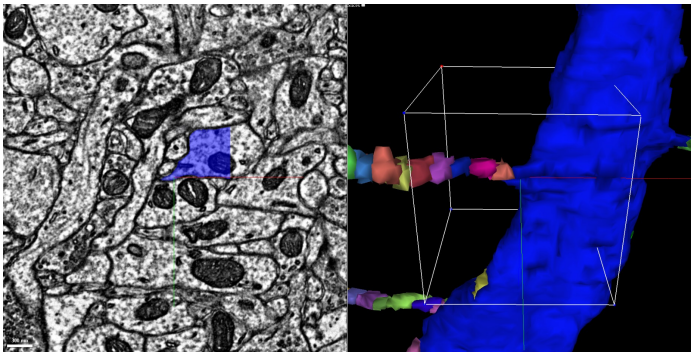


Figure 4: The supervoxels assembled in accordance with the output of the error correction network.

4 Putting it together

In this section, we describe how our error detector and error corrector are combined at test time. We begin with an initial segmentation whose remaining errors are assumed to be sparsely distributed. During the error correction phase, we iteratively update a segmentation represented as the connected components of a graph G whose vertices are segments in a strict over-segmentation (henceforth called supervoxels). We also maintain the combined error map associated with the current segmentation. We binarize the error map by thresholding it at 0.25.

Now we iteratively choose a location $\ell = (x, y, z)$ which has value 1 in the binarized combined error map. In a $P_x \times P_y \times P_z$ window centred on ℓ, we prepare an input for the error corrector by taking the union of all segments containing at least one white pixel in the error map. The error correction network produces from this input a binary image M representing the object containing the central pixel. For each supervoxel S touching the central $P_x/2 \times P_y/2 \times P_z/2$ window, let $M(S)$ denote the average value of M inside S. If $M(S) \notin [0.1, 0.9]$ for all S in the relevant window (i.e. the error corrector is confident in its prediction for each supervoxel), we add to G a clique on $\{S \mid M(S) > 0.9\}$ and delete from G all edges between $\{S \mid M(S) < 0.1\}$ and $\{S \mid M(S) > 0.9\}$. The effect of these updates is to change G to locally agree with M. Finally, we update the combined error map by applying the error detector at all locations where its decision could have changed.

We iterate until every location is zero in the error map or has been covered by a window at least twice by the error corrector. This stopping criterion guarantees that the algorithm terminates.

5 Experiments

5.1 Dataset

Our dataset is a sample of mouse primary visual cortex (V1) acquired using transmission electron microscopy at the Allen Institute for Brain Science. The voxel resolution is 3.6 nm $\times$ 3.6 nm $\times$ 40 nm.

A team of tracers produced multiple volumes of gold standard dense reconstruction, which amount to 530 Mvoxels. These volumes were used to train a neuronal boundary detection network. We applied the resulting boundary detector to a larger volume of size 4800 Mvoxels to produce a preliminary segmentation, which was then proofread by the tracers. This bootstrapped ground truth was used to train the error detector and corrector. A subvolume of size 910 Mvoxels was reserved for validation, and a subvolume of size 910 Mvoxels was reserved for testing.

Producing the gold standard segmentation required a total of ~ 560 tracer hours, while producing the bootstrapped ground truth required ~ 670 tracer hours.

5.2 Baseline segmentation

Our baseline segmentation was produced using a pipeline of multiscale convolutional networks for neuronal boundary detection, watershed, and mean affinity agglomeration [10]. We describe the pipeline in detail in Appendix A. The segmentation performance values reported for the baseline are taken at a mean affinity agglomeration threshold of 0.23, which minimizes the variation of information error metric [12, 15] on the test volumes.

5.3 Training procedures

Sampling procedure Here we describe our procedure for choosing a random point location in a segmentation. Uniformly random sampling is unsatisfactory since large objects such as dendritic shafts will be overrepresented. Instead, given a segmentation, we sample a location (x, y, z) with probability inversely proportional to the fraction of a window of size $128 \times 128 \times 16$ centred at (x, y, z) which is occupied by the object containing the central pixel.

Training of error detector An initial segmentation containing errors was produced using our baseline neuronal boundary detector combined with mean affinity agglomeration at a threshold of 0.3. Point locations were sampled according to the sampling procedure specified in 5.3. We augmented all of our data with rotations and reflections.

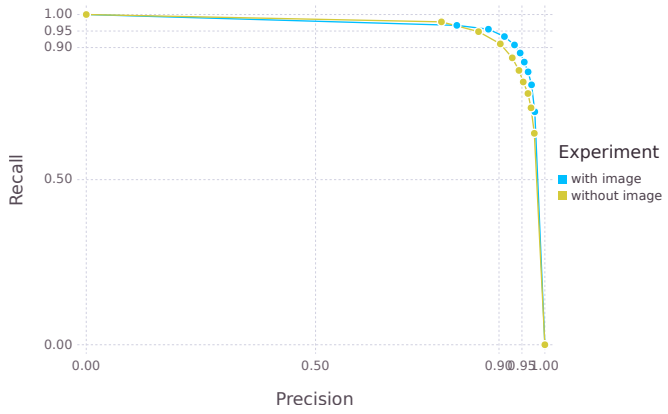


Figure 5: Precision and recall for error detection, both with and without access to the raw image. In the test volume, there are 8248 error free locations and 944 locations with errors. In practice, we use threshold which guarantees > 95% recall and > 85% precision.

5.4 Training of error corrector

We sampled locations in the ground truth segmentation as in 5.3. At each location $\ell = (x, y, z)$, we generated a training example as follows. Let Obj_ℓ be the ground truth object touching ℓ. We selected a random subset of the objects in the window centred on ℓ including Obj_ℓ. To be specific, we chose a number p uniformly at random from $[0, 1]$, and then selected each segment in the window with probability p in addition Obj_ℓ. The input is a binary mask representing the union of the selected objects along with the raw EM image, and the desired output is a binary mask representing only Obj_ℓ. The dataset was augmented with rotations, reflections, simulated misalignments and missing sections [**?**].

Note that this training procedure uses only the ground truth segmentation and is completely independent of the error detector and the baseline segmentataion. This convenient property is justified by the fact that if the error detector is perfect, the error corrector only ever receives as input unions of complete objects.

5.5 Error detection results

To measure the quality of error detection, we densely sampled points in our test volume as in Appendix 5.3. In order to remove ambiguity over the precise location of errors, we filtered out points which contained an error within a surrounding window of size $80 \times 80 \times 8$ but not a window of size $40 \times 40 \times 4$. These locations were all unique, in that two locations in the same object were separated by at least $80, 80, 8$ in x, y, z, respectively. Precision and recall simultaneously exceed 90% (see Figure 5). Empirically, many of the false positive examples come where a dendritic spine head curls back and touches its trunk. These examples locally appear to be incorrectly merged objects.

We trained one error detector with access to the raw image and one without. The network's admirable performance even without access to the image as seen in Figure 5 supports our hypothesis that error detection is a relatively easy task and can be performed using only shape cues.

5.6 Error correction results

In order to demonstrate the importance of error detection in error correction, ran two experiments: one in which the binary mask input to the error corrector was simply the union of all segments in the window ("without advice"), and one in which the binary mask was the union of all segments with a detected error ("with advice"). In the "without advice" mode, the network is essentially asked to reconstruct the object overlapping the central pixel in one shot. Table 1 shows that advice confers a considerable advantage in performance on the error corrector.

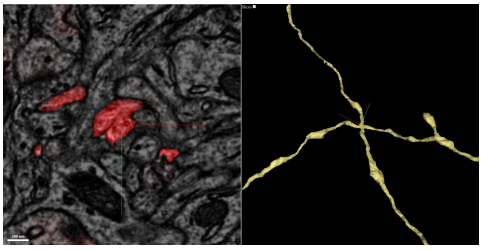


Figure 6: An example of a detected error. The right shows two incorrectly merged axons, and the left shows the predicted combined error map (defined in 2.1) overlaid on the corresponding 2D image in red.

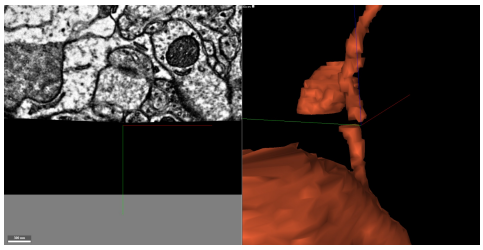


Figure 7: A difficult location with missing data in one section combined with a misalignment between sections. Our algorithm was able to correctly merge across the missing data.

Table 1: Comparing segmentation performance

	VI_{merge}	VI_{split}	Rand Recall	Rand Precision
Baseline	0.162	0.142	0.952	0.954
Without Advice	0.130	0.057	0.956	0.979
With Advice	**0.088**	**0.052**	**0.974**	**0.980**

It is sometimes difficult to assess the significance of an improvement in variation of information or rand score since changes can be dominated by modifications to a few large objects. Therefore, we decomposed the variation of information into a score for each object in the ground truth. Figure 8 summarizes the distribution of the values of $VI(i) = VI_{merge}(i) + VI_{split}(i)$ for all segments i in the ground truth. See Appendix E for a precise definition of $VI(i)$.

The number of errors from the set in 5.5 that were fixed or introduced by our iterative refinement procedure is shown in 2. Our iterative refinement procedure fixed a significant fraction of the remaining errors.

5.7 Cost analysis

Table 3 shows the cost of the most expensive parts of our segmentation pipeline. The combined cost of error detection and correction is within an order of magnitude of the cost of neuronal boundary detection. The selectivity of our error detection network allowed us to run error correction at roughly 10% of the possible locations in the image. Therefore, the cost of error detection is more than justified by the subsequent savings during the error correction phase. As the error rate of the initial

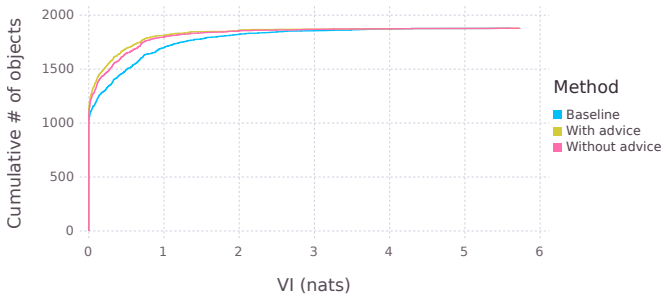


Figure 8: Per-object vi scores for the 940 reconstructed objects in our test volume. Almost 800 objects are completely error free in our segmentation. These objects are likely all axons; almost every dendrite has a couple of errors.

Table 2: Number of errors fixed and introduced relative to the baseline. These numbers should be taken with a grain of salt since topologically insignificant changes could result in errors.

	#Errors	#Errors fixed rel. baseline	#Errors introduced rel. baseline
Baseline	944	-	-
Without Advice	474	547	77
With Advice	**305**	707	68

segmentation decreases and the precision of the error detector increases, the number of locations requiring error correction will only fall further.

Table 3: Computation time for a volume of size $2048 \times 2048 \times 256$ using a single TitanX Pascal GPU

Boundary Detection	18 mins
Error Detection	25 mins
Error Correction	55 mins

6 Conclusion and future directions

We have developed a error detector for the neuronal segmentation problem and combined it with an error correction module. In particular, we have shown that our error detectors are able to exploit priors on neuron shape, having reasonable performance even without access to the raw image. We have made significant savings in computation by applying expensive error correction procedures only where predicted necessary by the error detector. Finally, we have demonstrated that the "advice" of error detection improves an error correcton module, improving segmentation performance upon our baseline.

We expect that significant improvements in the accuracy of error detection could come from aggressive data augmentation. We can mutilate a ground truth segmentation in arbitrary (or even adversarial) ways to produce unlimited examples of errors.

An error detection module has many potential uses beyond the ones presented here. For example, we could use error detection to direct ground truth annotation effort toward mistakes. If sufficiently accurate, it could also be used directly as a learning signal for segmentation algorithms on unlabelled data. The idea of co-training our error-correction and error-detection networks is natural in view of recent work on generative adversarial networks [14, 11].

Author contributions and acknowledgements

JZ conducted most of the experiments and evaluation. IT (along with Will Silversmith) created much of the infrastructure necessary for visualization and running our algorithms at scale. HSS secured funding and played an advisory role.

The dataset used for this project was acquired at the Allen Institute for Brain Science. The ground truth for this project was created by Ben Silverman, Merlin Moore, Sarah Morejohn, Selden Koolman, Ryan Willie, Kyle Willie, and Harrison Macgowan. Kisuk Lee trained the boundary detectors used to generate our baseline segmentation. We thank Kisuk Lee for several helpful conversations and Nico Kemnitz for proofreading a draft of this paper. We thank Jeremy Maitin-Shepard and the other contributors to the neuroglancer project for creating an invaluable visualization tool.

We thank Barton Fiske of NVIDIA Corporation for providing us with early access to Titan X Pascal GPU used in this research. This research was supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior/ Interior Business Center (DoI/IBC) contract number D16PC0005. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily

representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/IBC, or the U.S. Government.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

[2] Thorsten Beier, Constantin Pape, Nasim Rahaman, Timo Prange, Stuart Berg, Davi D Bock, Albert Cardona, Graham W Knott, Stephen M Plaza, Louis K Scheffer, et al. Multicut brings automated neurite segmentation closer to human performance. *Nature Methods*, 14(2):101–102, 2017.

[3] Jan Funke, Fabian David Tschopp, William Grisaitis, Chandan Singh, Stephan Saalfeld, and Srinivas C Turaga. A deep structured learning approach towards automating connectome reconstruction from 3d electron micrographs. *arXiv preprint arXiv:1709.02974*, 2017.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

[5] Viren Jain, Joseph F Murray, Fabian Roth, Srinivas Turaga, Valentin Zhigulin, Kevin L Briggman, Moritz N Helmstaedter, Winfried Denk, and H Sebastian Seung. Supervised learning of image restoration with convolutional networks. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[6] Viren Jain, Srinivas C. Turaga, Kevin L. Briggman, Moritz Helmstaedter, Winfried Denk, and H. Sebastian Seung. Learning to agglomerate superpixel hierarchies. In *NIPS*, 2011.

[7] Michał Januszewski, Jörgen Kornfeld, Peter H Li, Art Pope, Tim Blakely, Larry Lindsey, Jeremy B Maitin-Shepard, Mike Tyka, Winfried Denk, and Viren Jain. High-precision automated reconstruction of neurons with flood-filling networks. *bioRxiv*, page 200675, 2017.

[8] Michał Januszewski, Jeremy Maitin-Shepard, Peter Li, Jörgen Kornfeld, Winfried Denk, and Viren Jain. Flood-filling networks, Nov 2016. URL `https://arxiv.org/abs/1611.00421`.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, Jan 2017. URL `https://arxiv.org/abs/1412.6980`.

[10] Kisuk Lee, Jonathan Zung, Peter Li, Viren Jain, and H. Sebastian Seung. Superhuman accuracy on the SNEMI3D connectomics challenge. *CoRR*, abs/1706.00120, 2017. URL `http://arxiv.org/abs/1706.00120`.

[11] Pauline Luc, Camille Couprie, Soumith Chintala, and Jakob Verbeek. Semantic segmentation using adversarial networks. *CoRR*, abs/1611.08408, 2016. URL `http://arxiv.org/abs/1611.08408`.

[12] Marina Meilă. Comparing clusterings—an information based distance. *Journal of Multivariate Analysis*, 98(5):873 – 895, 2007. ISSN 0047-259X. doi: http://dx.doi.org/10.1016/j.jmva.2006.11.013. URL `http://www.sciencedirect.com/science/article/pii/S0047259X06002016`.

[13] Yaron Meirovitch, Alexander Matveev, Hayk Saribekyan, David Budden, David Rolnick, Gergely Odor, Seymour Knowles-Barley, Thouis Raymond Jones, Hanspeter Pfister, Jeff William Lichtman, and Nir Shavit. A multi-pass approach to large-scale connectomics, 2016.

[14] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014. URL `http://arxiv.org/abs/1411.1784`.

[15] Juan Nunez-Iglesias, Ryan Kennedy, Toufiq Parag, Jianbo Shi, and Dmitri B. Chklovskii. Machine Learning of Hierarchical Clustering to Segment 2D and 3D Images. *PLOS ONE*, 8

(8):1–11, 08 2013. doi: 10.1371/journal.pone.0071715. URL https://doi.org/10.1371/journal.pone.0071715.

[16] Juan Nunez-Iglesias, Ryan Kennedy, Stephen M. Plaza, Anirban Chakraborty, and William T. Katz. Graph-based active learning of agglomeration (gala): a python library to segment 2d and 3d neuroimages, 2014. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3983515/.

[17] Pedro O. Pinheiro, Ronan Collobert, and Piotr Dollár. Learning to segment object candidates. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 1990–1998, Cambridge, MA, USA, 2015. MIT Press. URL http://dl.acm.org/citation.cfm?id=2969442.2969462.

[18] Mengye Ren and Richard S. Zemel. End-to-end instance segmentation and counting with recurrent attention. *CoRR*, abs/1605.09410, 2016. URL http://arxiv.org/abs/1605.09410.

[19] David Rolnick, Yaron Meirovitch, Toufiq Parag, Hanspeter Pfister, Viren Jain, Jeff W. Lichtman, Edward S. Boyden, and Nir Shavit. Morphological error detection in 3d segmentations. *CoRR*, abs/1705.10882, 2017. URL http://arxiv.org/abs/1705.10882.

[20] Bernardino Romera-Paredes and Philip H. S. Torr. Recurrent instance segmentation. *CoRR*, abs/1511.08250, 2015. URL http://arxiv.org/abs/1511.08250.

[21] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, May 2015. URL https://arxiv.org/abs/1505.04597.

[22] Helene Schmidt, Anjali Gour, Jakob Straehle, Kevin M Boergens, Michael Brecht, and Moritz Helmstaedter. Axonal synapse sorting in medial entorhinal cortex. *Nature*, 549(7673):469, 2017.

[23] 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., Feb 2010. URL https://www.ncbi.nlm.nih.gov/pubmed/19922289.

[24] John G White, Eileen Southgate, J Nichol Thomson, and Sydney Brenner. The structure of the nervous system of the nematode caenorhabditis elegans: the mind of a worm. *Phil. Trans. R. Soc. Lond*, 314:1–340, 1986.

[25] Tao Zeng, Bian Wu, and Shuiwang Ji. Deepem3d: approaching human-level performance on 3d anisotropic em image segmentation. *Bioinformatics*, 33(16):2555–2562, 2017. doi: 10.1093/bioinformatics/btx188. URL +http://dx.doi.org/10.1093/bioinformatics/btx188.

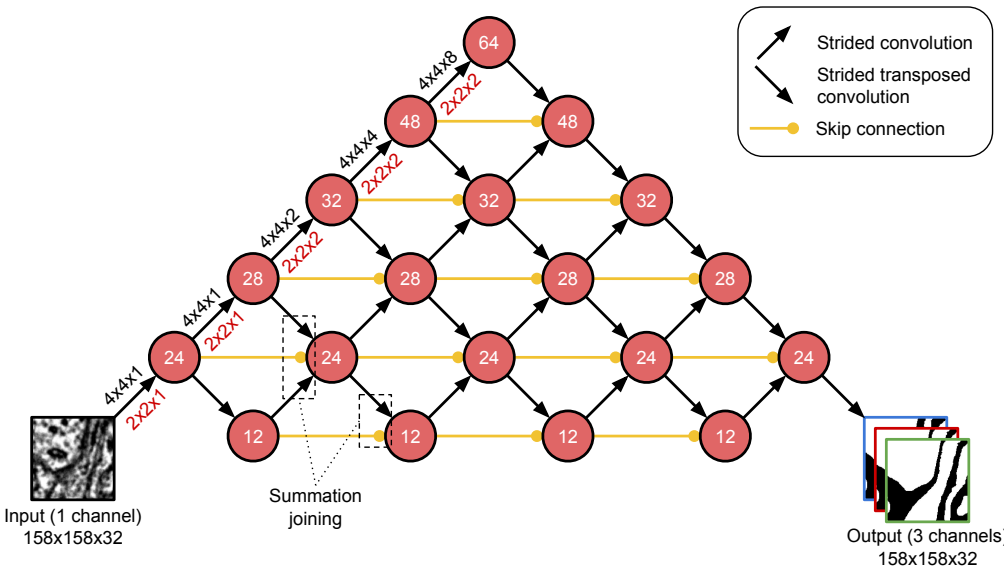


Figure 9: Architecture for the baseline neuronal boundary detection. Each node represents a layer and the number inside represents the number of feature maps. The layers closer to the top of the diagram have lower resolution than the layers near the bottom. The diagonal arrows represent strided convolutions, while the horizontal arrows represent skip connections.

Appendices

A Baseline Neuronal Boundary Detection

In this section we describe

A.1 Architecture

A.2 Dataset

A.3 Training

A.4 Postprocessing

A.5 Inference

B Network Architectures

Due to the anisotropy of the resolution of the images in our dataset, we design our networks so that the first convolutions are exclusively 2D while later convolutions are 3D. The field of view of a unit in the higher layers is therefore roughly cubic.

To limit the number of parameters in our model, we factorize all 3D convolutions into a 2D convolution followed by a 1d convolution in z. We also use weight sharing between some convolutions at the same height.

C Sampling Procedure

In this section, we describe our procedure for choosing a random point location in a segmentation. Uniformly random sampling is unsatisfactory since large objects such as dendritic trunks will be overrepresented. Instead, given a segmentation, we sample a location $\ell = (x, y, z)$ with probability

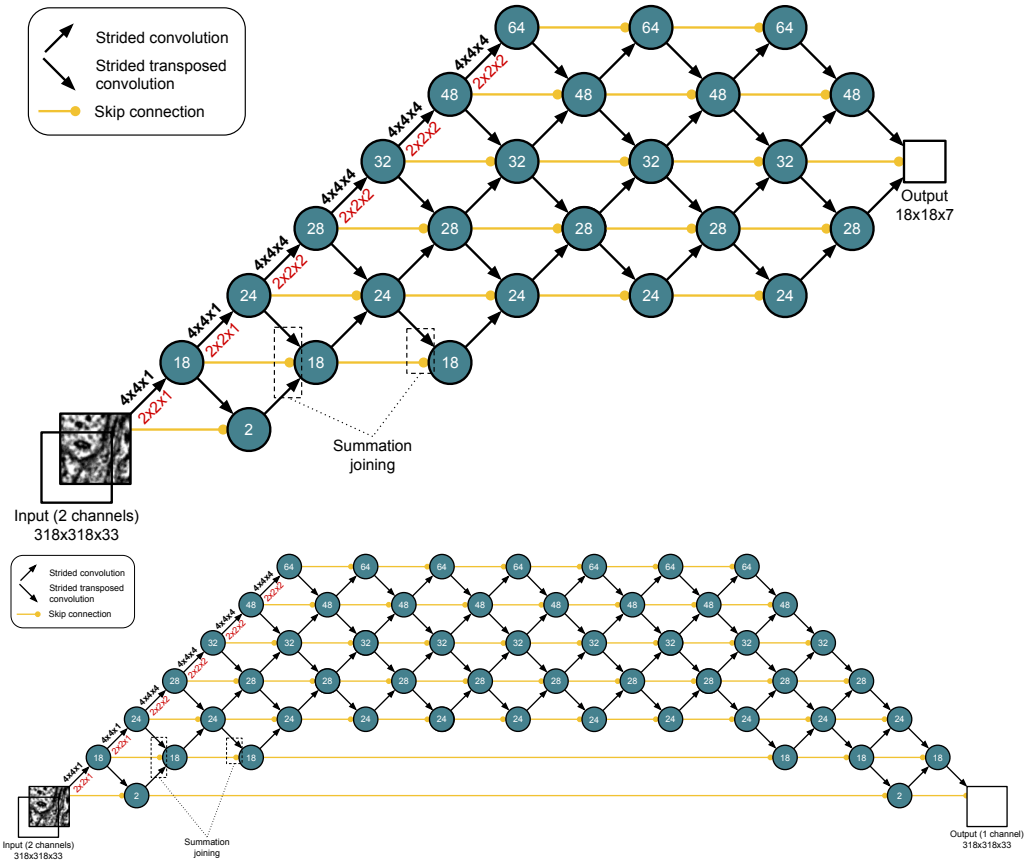


Figure 10: Architectures for the error detection and error correction modules respectively. Each node represents a layer and the number inside represents the number of feature maps. The layers closer to the top of the diagram have lower resolution than the layers near the bottom. We make savings in computation by minimizing the number of high resolution feature maps. The diagonal arrows represent strided convolutions, while the horizontal arrows represent skip connections.

inversely proportional to the fraction of a window of size $128 \times 128 \times 16$ centred at ℓ which is occupied by the object containing pixel ℓ.

D Training Details

The neural networks were implemented in TensorFlow [1] and trained using 4 TitanX Pascal GPUs with synchronous gradient descent. We used the Adam optimizer [9]. Both networks were trained until the loss on a validation set plateaued. The error detection network trained for 700,000 iterations (approximately one week), while the error correction network trained for 1,700,000 iterations (approximately three weeks).

E Per-Object VI Score

Recall that the variation of information between two segmentations may be computed as

$$VI_{split} = -\frac{1}{\sum_{i,j} r_{ij}} \sum_{i,j} r_{ij} \log(r_{ij}/p_i)$$

$$VI_{merge} = -\frac{1}{\sum_{i,j} r_{ij}} \sum_{i,j} r_{ij} \log(r_{ij}/q_j)$$

$$p_i = \sum_j r_{ij}$$

$$q_j = \sum_i r_{ij}$$

where r_{ij} is the number of pixels in common between the i^{th} segment of the ground truth segmentation and the j^{th} segment of the proposed segmentation [15].

We define the split and merge scores for ground truth segment i as

$$VI_{split}(i) = -\sum_j r_{ij}/p_i \log(r_{ij}/p_i)$$

$$VI_{merge}(i) = -\sum_j r_{ij}/p_i \log(r_{ij}/q_j)$$

Both quantities have units of bits. $VI_{split}(i)$ is zero iff ground truth segment i is contained within a segment in the proposed segmentation, while $VI_{merge}(i)$ is zero iff ground truth segment i is the union of one or more segments in the proposed segmentation. The total score VI_{split} or VI_{merge} is a weighted sum of the per-object scores $VI_{split}(i)$, $VI_{merge}(i)$ respectively.

F Training Details

The neural networks were implemented in TensorFlow [1] and trained using 4 TitanX Pascal GPUs with synchronous gradient descent. We used the Adam optimizer [9]. Both networks were trained until the loss on a validation set plateaued. The error detection network trained for 700,000 iterations (approximately one week), while the error-correcting network trained for 1,700,000 iterations (approximately three weeks).