An Error Detection and Correction Framework for Connectomics

Jonathan Zung
Princeton University
`jzung@princeton.edu`

Ignacio Tartavull[*]
Princeton University
`tartavull@princeton.edu`

Kisuk Lee[†]
MIT
`kisuklee@mit.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 image segmentation more generally. Both tasks take as input the raw image and a binary mask representing a candidate object. For the error detection task, the desired output is a map of split and merge errors in the object. For the error correction task, the desired output is the true object. We call this object mask pruning, because the candidate object mask is assumed to be a superset of the true object. We train multiscale 3D convolutional networks to perform both tasks. We find that the error-detecting net can achieve high accuracy. The accuracy of the error-correcting net is enhanced if its input object mask is "advice" (union of erroneous objects) from the error-detecting net.

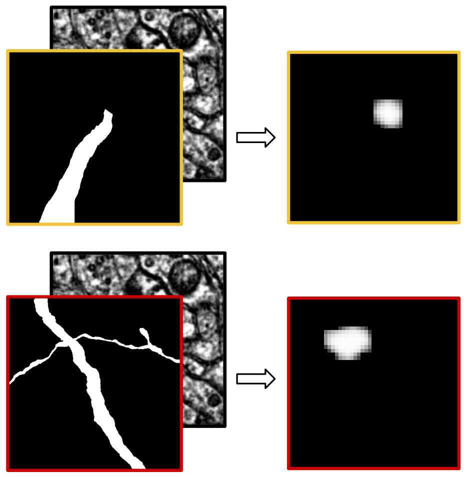
1 Introduction

While neuronal circuits can be reconstructed from volumetric electron microscopic imagery, the process has historically [39] and even recently [37] 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 networks were first applied to neuronal boundary detection a decade ago [14, 38]. Since then convolutional nets have become the standard approach, and the accuracy of boundary detection has become impressively high [40, 3, 21, 9].

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 [25] or learned [33] computations. However, a typical segmentation contains both split and merge errors, so it would be desirable to include both in the error detection task.

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*

[*]Current address: Uber Advanced Technology Group
[†]Co-affiliation: Princeton University


(a) Error detection task for split (top) and merge (bottom) errors. The desired output is an error map. A voxel in the error map is lit up if and only if a window centered on it contains a split or merge error. We also consider a variant of the task in which the object mask is the sole input.

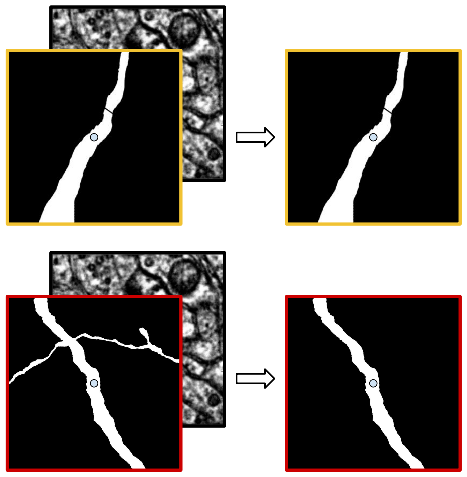

(b) The object mask pruning task. The input mask is assumed to be a superset of a true object. The desired output (right) is the true object containing the central voxel (center circle). In the first case there is nothing to prune, while in the second case the object not overlapping the central voxel is erased.

Figure 1: Error detection and correction tasks. For both tasks, the inputs are a candidate object mask and the original image. Note that diagrams are 2D for illustrative purposes, but in reality the inputs and outputs are 3D.

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 [15, 28]. 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 [17, 16] or *MaskExtend* [25], each iteration of which is the *addition* of voxels to an object to create a true object. Iterative mask extension has been studied in other work on instance segmentation in computer vision [34, 32]. The task of generating an object mask *de novo* from an image has also been studied in computer vision [30].

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-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 show that the error-detecting net can provide "advice" to the error-correcting net 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. The idea of using the error detector to choose locations for the error corrector was proposed previously though not actually implemented [25]. Furthermore, the idea of using the error detector to not only choose locations but provide "advice" is novel as far as we know.

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-detecting net 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 parts from the candidate mask described above. This contrasts with the flood-filling task which involves an unconstrained search for new parts to add. Given that most voxels are *not* a part of the object to be reconstructed, an upper bound on the object is usually more informative than a lower bound. As an added benefit, selective application of the error correction module near likely errors makes efficient use of our computational budget [25].

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.

Some of the design choices we made in our neural networks may be of interest to other researchers. Our error-correcting net is trained to produce a vector field via metric learning instead of directly producing an object mask. The vector field resembles a semantic labeling of the image, so this approach blurs the distinction between instance and semantic segmentation. This idea is relatively new in computer vision [10, 7, 5]. Our multiscale convolutional net architecture, while similar in spirit to the popular U-Net [35], has some novelty. With proper weight sharing, our model can be viewed as a feedback recurrent convolutional net unrolled in time (see the appendix for details). Although our model architecture is closely related to the independent works of [36, 13, 8], we contribute a *feedback recurrent* convolutional net interpretation.

2 Error detection

2.1 Task specification: detecting split and merge errors

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 voxel 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 voxel-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.

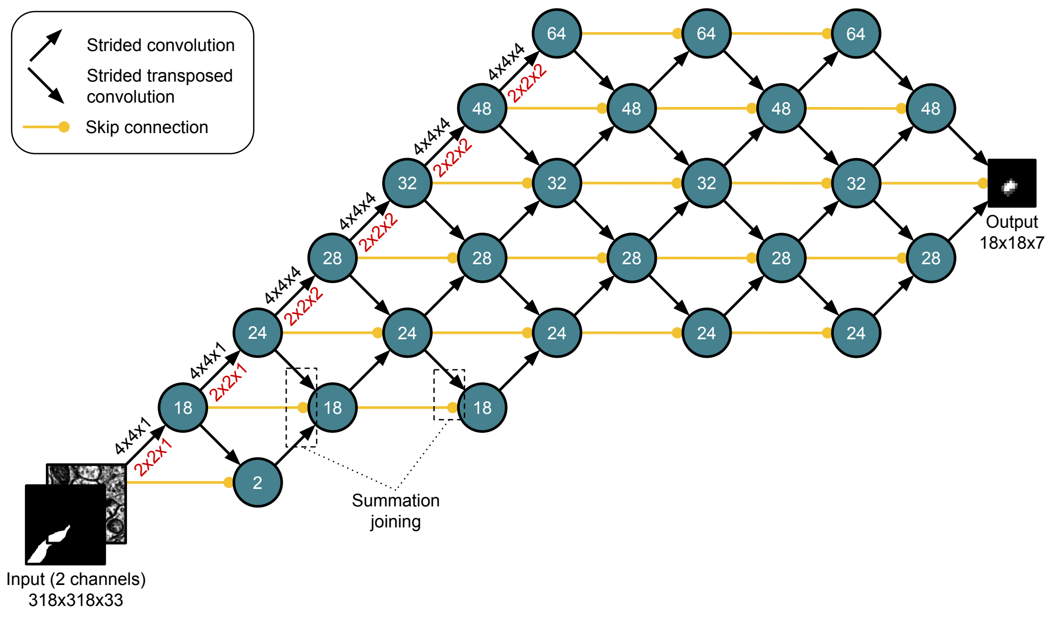
We could use a less stringent measure than voxel-wise equality that disregards small perturbations of the boundaries of objects. However, our proposed segmentations are all composed of the same building blocks (supervoxels) as the ground truth segmentation, so this is not an issue for us.

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 Architecture of the error-detecting net

We take a fully supervised approach to error detection. We implement error detection using a multiscale 3D convolutional network. The architecture is detailed in Figure 2. Its design is informed by experience with convolutional nets for neuronal boundary detection [21] and reflects recent trends in neural network design [35, 12]. 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) $Err_{46 \times 46 \times 7}$. 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.



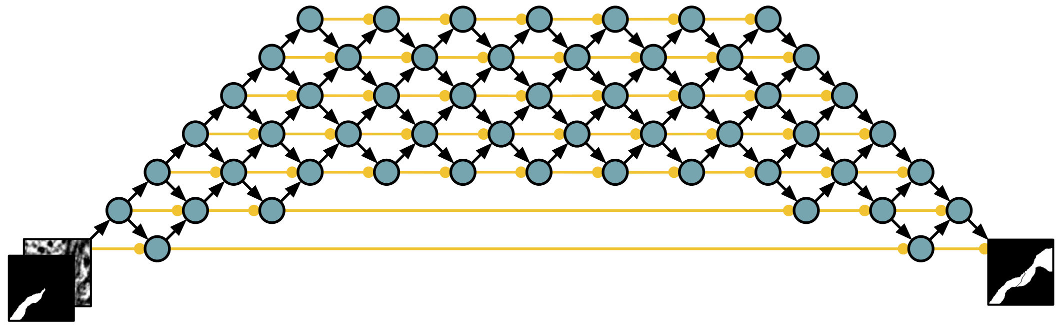


Figure 2: Architectures for the error-detecting and error-correcting nets respectively. Each node represents a layer and the number inside represents the number of feature maps. The layers closer to the top of the diagram have lower resolution than the layers near the bottom. We make savings in computation by minimizing the number of high resolution feature maps. The diagonal arrows represent strided convolutions, while the horizontal arrows represent skip connections. Associated with the diagonal arrows, black numbers indicate filter size and red numbers indicate strides in x, y and z dimension. Due to the anisotropy of the resolution of the images in our dataset, we design our nets 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-dimension. We also use weight sharing between some convolutions at the same height. Note that the error-correcting net is a prolonged, symmetric version of the error-detecting net. For more detail of the error corrector, see the appendix.

3 Error correction

3.1 Task specification: object mask pruning

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 true object overlapping the central voxel. The candidate object mask is assumed to be a superset of the true object.

3.2 Architecture of the error-correcting net

Yet again, we implement error correction using a multiscale 3D convolutional network. The architecture is detailed in Figure 2. One difficulty with training a neural network to reconstruct the object containing the central voxel is that the desired output can change drastically as the central voxel moves between objects. We use an intermediate representation whose role is to soften this dependence on the location of the central voxel. The desired intermediate representation is a $k = 6$ 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 voxel as follows:

$$M(x, y, z) = \exp\left(-||v(x, y, z) - v(0, 0, 0)||^2\right),$$

where $(0, 0, 0)$ is the central voxel. When an over-segmentation is available, we replace $v(0, 0, 0)$ with the average of v over the supervoxel containing the central voxel. This trick makes it unnecessary to centre our windows far away from a boundary, as was necessary in [17]. 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.

4 How the error detector can "advise" the error corrector

Suppose that we would like to correct the errors in a baseline segmentation. Obviously, the error-detecting net can be used to find locations where the error-correcting net can be applied [25]. Less obviously, the error-detecting net can be used to construct the object mask that is the input to the error-correcting net. We refer to this object mask as the "advice mask" and its construction is important because the baseline object to be corrected might contain split as well as merge errors, while the object mask pruning task can correct only merge errors.

The advice mask is defined as the union of the baseline object at the central pixel with all other baseline objects in the window that contain errors as judged by the error-detecting net. The advice mask is a superset of the true object overlapping the central voxel, assuming that the error-detecting net makes no mistakes. Therefore advice is suitable as an input to the object mask pruning task.

The details of the above procedure are as follows. We begin with an initial baseline 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 voxel in the error map. The error correction network produces from this input a binary image M representing the object containing the central voxel. 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 $t = 2$ times by the error corrector. This stopping criterion guarantees that the algorithm terminates. In

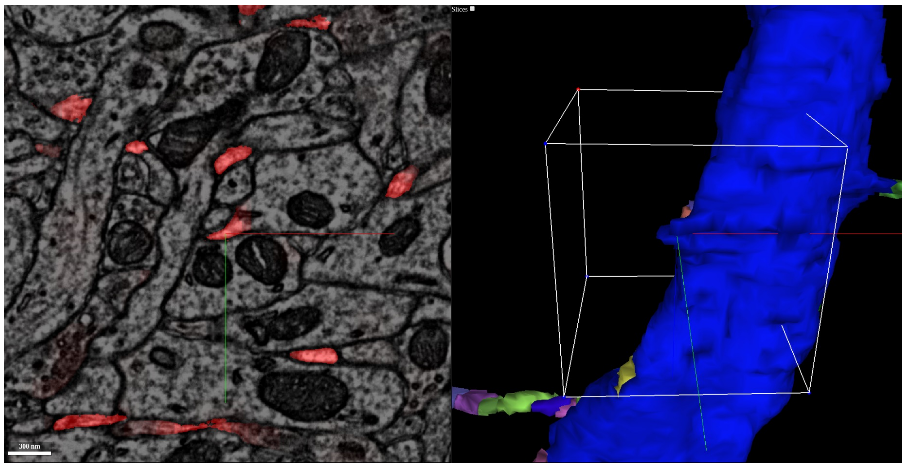


Figure 3: An example of a mistake in the initial segmentation. The dendrite is missing a spine. The red overlay on the left shows the combined error map (defined in Section 2.1); the stump in the centre of the image was clearly marked as an error.

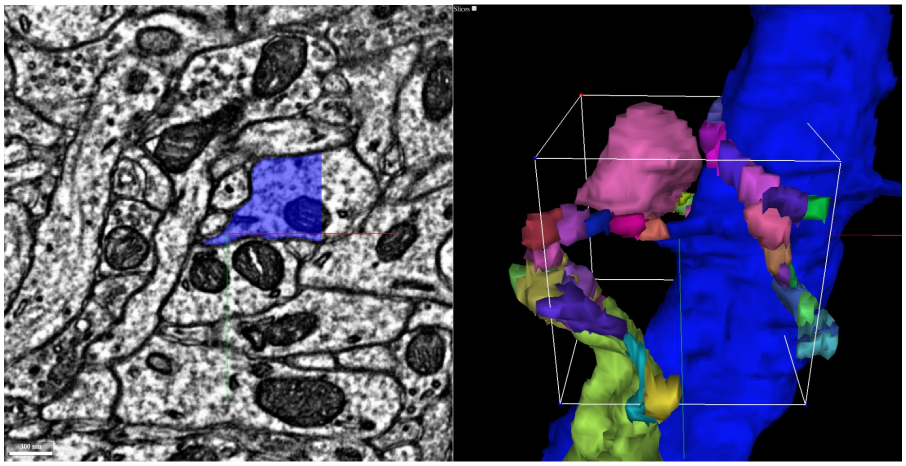


Figure 4: The right shows all objects which contained a detected error in the vicinity. For clarity, each supervoxel was drawn with a different colour. The union of these objects is the binary mask which is provided as input to the error correction network. For clarity, these objects were clipped to 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.

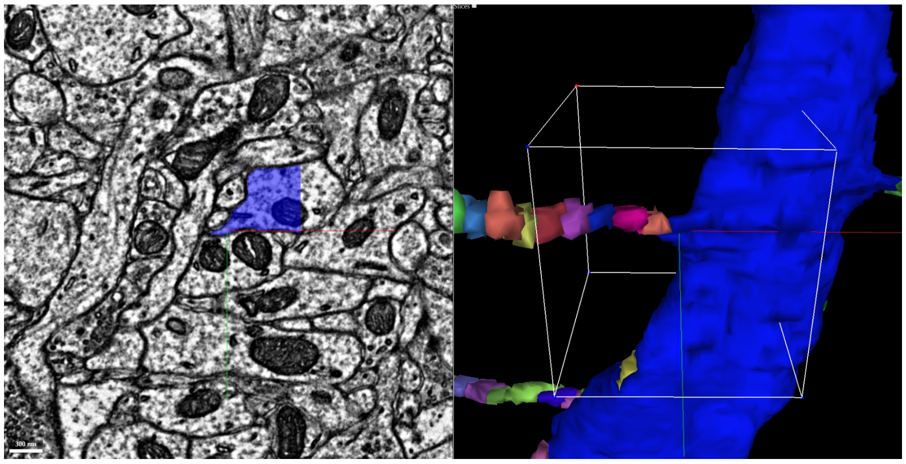


Figure 5: The supervoxels assembled in accordance with the output of the error correction network.

practice, the segmentation converges without this auxiliary stopping condition to a state in which the error corrector fails confidence threshold everywhere. However, it is hard to certify convergence since it is possible that the error corrector could give different outputs on slightly shifted windows. Based on our validation set, increasing t beyond 2 did not measurably improve performance.

Note that this algorithm deals with split and merge errors, but cannot fix errors already present at the supervoxel level.

5 Experiments

5.1 Dataset

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

Human experts used the VAST software tool [19, 4] to densely reconstruct multiple volumes that amounted to 530 Mvoxels of ground truth annotation. These volumes were used to train a neuronal boundary detection network (see the appendix for architecture). 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 [21]. We describe the pipeline in detail in the appendix. 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 [24, 27] 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 voxel [17].

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 above. We augmented all of our data with rotations and reflections. We used a pixelwise cross-entropy loss.

Training of error corrector We sampled locations in the ground truth segmentation as described above. 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 at ℓ was then a binary mask representing the union of the selected objects along with the raw EM image, and the desired output was a binary mask representing only Obj_ℓ. The dataset was augmented with rotations, reflections, simulated misalignments and missing sections [21]. We used a pixelwise cross-entropy loss.

Note that this training procedure uses only the ground truth segmentation and is completely independent of the error detector and the baseline segmentation. This convenient property is justified by

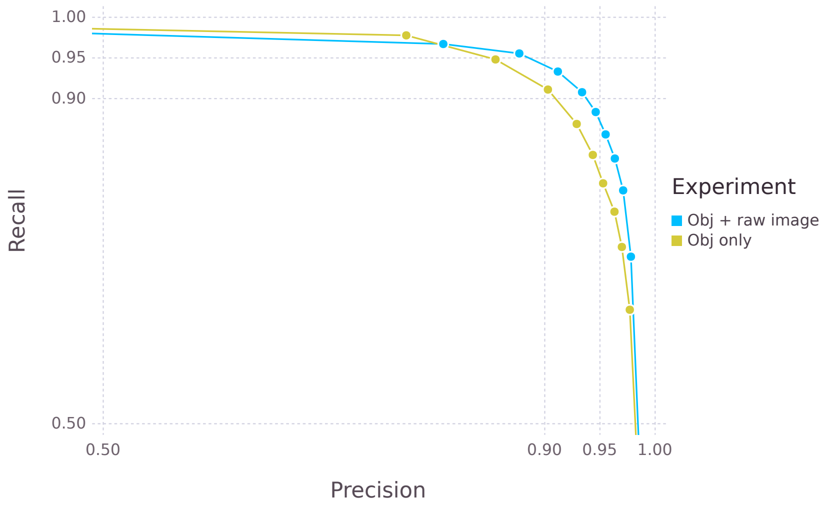


Figure 6: Precision and recall for error detection, both with and without access to the raw image. In the test volume, there are 8248 error free locations and 944 locations with errors. In practice, we use threshold which guarantees $> 95\%$ recall and $> 85\%$ precision.

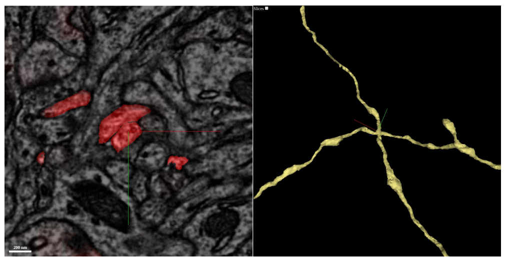


Figure 7: An example of a detected error. The right shows two incorrectly merged axons, and the left shows the predicted combined error map (defined in Section 2.1) overlaid on the corresponding 2D image in red.

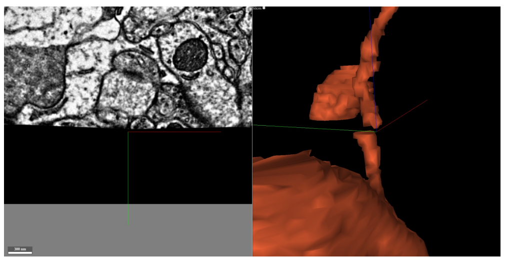


Figure 8: A difficult location with missing data in one section combined with a misalignment between sections. The error-correcting net was able to trace across the missing data.

the fact that if the error detector is perfect, the error corrector only ever receives as input unions of complete objects.

5.4 Error detection results

To measure the quality of error detection, we densely sampled points in our test volume as in Section 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% (Figure 6). Empirically, many of the false positive examples come from 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 6 supports our hypothesis that error detection is a relatively easy task and can be performed using only shape cues.

Merge errors qualitatively appear to be especially easy for the network to detect; an example is shown in Figure 7.

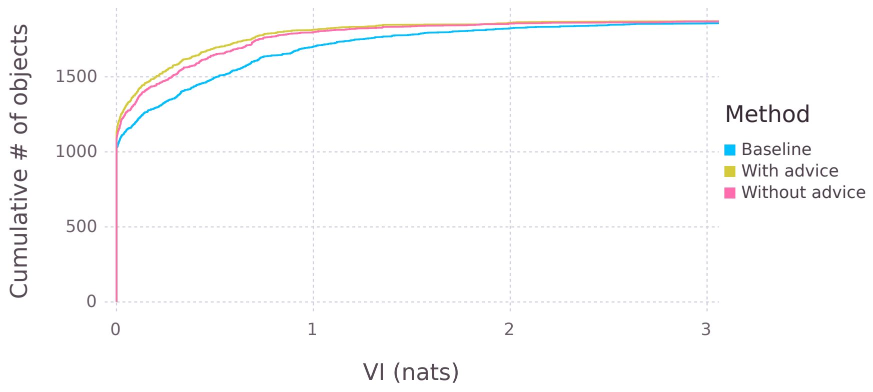


Figure 9: Per-object VI scores for the 940 reconstructed objects in our test volume. Almost 800 objects are completely error free in our segmentation. These objects are likely all axons; almost every dendrite is missing a few spines.

5.5 Error correction results

In order to demonstrate the importance of error detection to error correction, we 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 voxel in one shot. Table 1 shows that advice confers a considerable advantage in performance on the error corrector.

It is sometimes difficult to assess the significance of an improvement in the variation of information or *Rand score* [31, 2] 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 9 summarizes the cumulative distribution of the values of $VI(i) = VI_{\text{merge}}(i) + VI_{\text{split}}(i)$ for all segments i in the ground truth. See the appendix for a precise definition of $VI(i)$.

The number of errors from the set in Section 5.4 that were fixed or introduced by our iterative refinement procedure is shown in Figure 2. These numbers should be taken with a grain of salt since topologically insignificant changes could result in errors. Regardless, it is clear that our iterative refinement procedure fixed a significant fraction of the remaining errors and that "advice" improves the error corrector.

The results are qualitatively impressive as well. The error-correcting network is sometimes able to correctly merge disconnected objects, as exemplified in Figure 8.

5.6 Computational cost analysis

Table 3 shows the computational cost of the most expensive parts of our segmentation pipeline. Boundary detection and error detection are run on the entire image, while error correction is run on roughly 10% of the possible locations in the image. Error correction is still the most costly step, but it would be $10\times$ more costly without restricting to the locations found by the error-detecting network. Therefore, the cost of error detection is more than justified by the subsequent savings during the

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**

Table 2: Number of errors fixed and introduced relative to the baseline

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

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

Boundary detection	Error detection	Error correction
18 mins	25 mins	55 mins

error correction phase. The number of locations requiring error correction will fall even further if the precision of the error detector increases or the error rate of the initial segmentation decreases.

6 Conclusion and future directions

We have developed an 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 correction 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-detecting and error-correcting nets is natural in view of recent work on generative adversarial networks [26, 23].

Author contributions & acknowledgements

JZ conceptualized the study and 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. KL produced the baseline segmentation. HSS helped with the writing.

We are grateful to Clay Reid, Nuno da Costa, Agnes Bodor, Adam Bleckert, Dan Bumbarger, Derrick Britain, JoAnn Buchannan, and Marc Takeno for acquiring the TEM dataset at the Allen Institute for Brain Science. The ground truth annotation was created by Ben Silverman, Merlin Moore, Sarah Morejohn, Selden Koolman, Ryan Willie, Kyle Willie, and Harrison MacGowan. We thank Nico Kemnitz for proofreading a draft of this paper. We thank Jeremy Maitin-Shepard at Google and the other contributors to the Neuroglancer project for creating an invaluable visualization tool.

We acknowledge NVIDIA Corporation for providing us with early access to Titan X Pascal GPU used in this research, and Amazon for assistance through an AWS Research Grant. This research was supported by the Mathers Foundation, the Samsung Scholarship and 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] Ignacio Arganda-Carreras et al. Crowdsourcing the creation of image segmentation algorithms for connectomics. *Frontiers in Neuroanatomy*, 9:142, 2015. ISSN 1662-5129. doi: 10.3389/fnana.2015.00142. URL `http://journal.frontiersin.org/article/10.3389/fnana.2015.00142`.

[3] 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.

[4] Daniel Berger. *VAST Lite*. URL `https://software.rc.fas.harvard.edu/lichtman/vast/`.

[5] Bert De Brabandere, Davy Neven, and Luc Van Gool. Semantic instance segmentation with a discriminative loss function. *CoRR*, abs/1708.02551, 2017. URL `http://arxiv.org/abs/1708.02551`.

[6] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *CoRR*, abs/1511.07289, 2015. URL `http://arxiv.org/abs/1511.07289`.

[7] Alireza Fathi, Zbigniew Wojna, Vivek Rathod, Peng Wang, Hyun Oh Song, Sergio Guadarrama, and Kevin P. Murphy. Semantic instance segmentation via deep metric learning. *CoRR*, abs/1703.10277, 2017. URL `http://arxiv.org/abs/1703.10277`.

[8] Damien Fourure, Rémi Emonet, Élisa Fromont, Damien Muselet, Alain Trémeau, and Christian Wolf. Residual conv-deconv grid network for semantic segmentation. *CoRR*, abs/1707.07958, 2017. URL `http://arxiv.org/abs/1707.07958`.

[9] 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.

[10] Adam W. Harley, Konstantinos G. Derpanis, and Iasonas Kokkinos. Learning dense convolutional embeddings for semantic segmentation. *CoRR*, abs/1511.04377, 2015. URL `http://arxiv.org/abs/1511.04377`.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *CoRR*, abs/1502.01852, 2015. URL `http://arxiv.org/abs/1502.01852`.

[12] 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`.

[13] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q. Weinberger. Multi-scale dense convolutional networks for efficient prediction. *CoRR*, abs/1703.09844, 2017. URL `http://arxiv.org/abs/1703.09844`.

[14] 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.

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

[16] 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.

[17] 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`.

[18] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[19] Narayanan Kasthuri, Kenneth Jeffrey Hayworth, Daniel Raimund Berger, Richard Lee Schalek, José Angel Conchello, Seymour Knowles-Barley, Dongil Lee, Amelio Vázquez-Reina, Verena Kaynig, Thouis Raymond Jones, et al. Saturated reconstruction of a volume of neocortex. *Cell*, 162(3):648–661, 2015.

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

[21] 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`.

[22] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017.

[23] 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`.

[24] 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`.

[25] 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.

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

[27] 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`.

[28] 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/`.

[29] Pedro H. O. Pinheiro, Tsung-Yi Lin, Ronan Collobert, and Piotr Dollár. Learning to Refine Object Segments. *CoRR*, abs/1603.08695, 2016. URL `http://arxiv.org/abs/1603.08695`.

[30] 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`.

[31] William M. Rand. Objective Criteria for the Evaluation of Clustering Methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971. doi: 10.1080/01621459.1971.10482356. URL `http://www.tandfonline.com/doi/abs/10.1080/01621459.1971.10482356`.

[32] 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`.

[33] 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`.

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

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

[36] Shreyas Saxena and Jakob Verbeek. Convolutional Neural Fabrics. *CoRR*, abs/1606.02492, 2016. URL `http://arxiv.org/abs/1606.02492`.

[37] 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.

[38] S C Turaga, J F Murray, V Jain, F Roth, M Helmstaedter, K Briggman, W Denk, and H S Seung. Convolutional networks can learn to generate affinity graphs for image segmentation., Feb 2010. URL `https://www.ncbi.nlm.nih.gov/pubmed/19922289`.

[39] 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.

[40] 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`.

Supplementary Information for "An Error Detection and Correction Framework for Connectomics" by Zung et al. (*NIPS 2017*)

A Baseline neuronal boundary detection

In this section, we describe our baseline segmentation pipeline, which is similar to what is described in [21]. The major difference is our novel densely multiscale 3D convolutional network architecture for neuronal boundary detection, which is described below. (The same *class* of architecture was employed in error detection and error correction. See main text.)

A.1 Network architecture

Our proposed densely multiscale 3D convolutional network for neuronal boundary detection is illustrated in Figure S10. Our model is built upon U-Net [35] with several interesting architectural augmentations. Our model can be viewed as a pyramidal stack of the basic computational module (diamond-shaped box in Figure S10). This diamond-shaped module can be interpreted as a residual building block (see Figure 2 in [12]) with two residual pathways, one top-down and the other bottom-up. Thus our model is *fully residual* in the sense that every computational pathway involving horizontal information flow is passing through the residual modules. Moreover, every residual module refines its input representation by integrating both top-down and bottom-up information, thus allowing for *dense* intermixing of multiscale features. Our model's *dense* and *fully residual* architecture allows an incremental and iterative top-down/bottom-up refinement of internal representation, which is in contrast to U-Net and variants' more restricted coarse-to-fine top-down refinement [35, 29, 22].

From a different point of view, Figure S11 illustrates another important motivation for our densely multiscale convolutional net architecture. Our model can be viewed as a feedback recurrent convolutional network unrolled in time (Figure S11). Weight sharing across time makes our model exactly equivalent to a convolutional net with recurrent feedback connections unfolded through time, and this novel perspective provides a better framework for understanding one of the unique characteristics

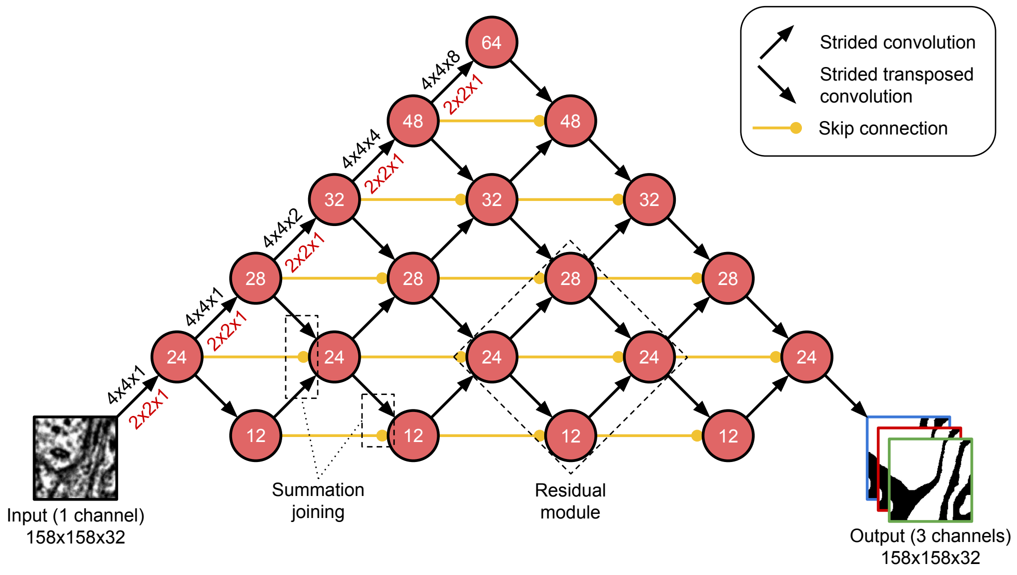


Figure S10: Architecture for the baseline neuronal boundary detection. Each node represents a layer and the number inside represents the number of feature maps. The layers closer to the top of the diagram have lower resolution than the layers near the bottom. The diagonal arrows represent strided convolutions, while the horizontal arrows represent skip connections. Associated with the diagonal arrows, the numbers indicate filter size (black) and strides (red) in x, y, and z-dimension. The target for our boundary detection network is a 3D *affinity graph* [38, 21, 9], thus outputting three channels corresponding to x (green), y (red), and z (blue) affinity maps, respectively.

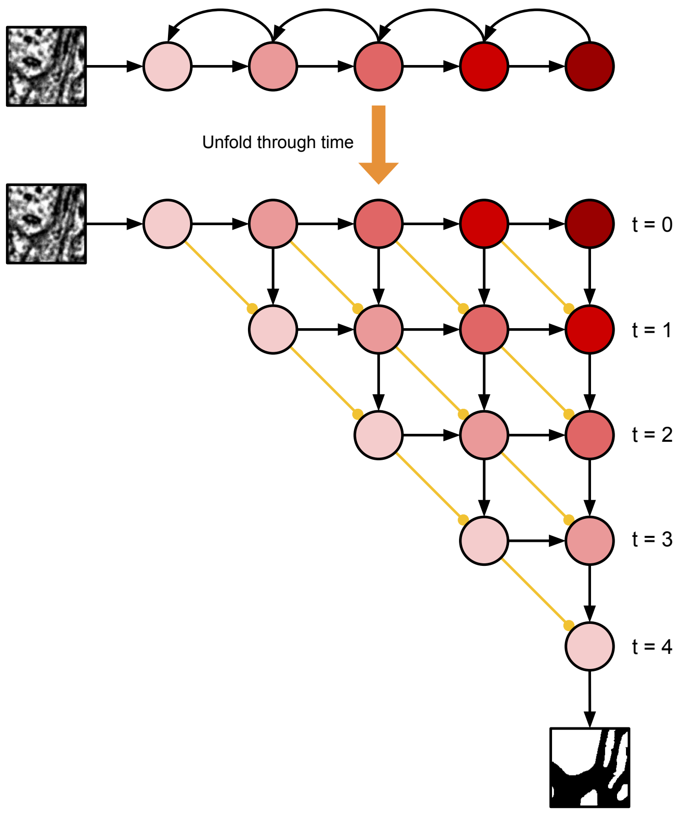


Figure S11: Feedback recurrent convolutional network unrolled in time (see Section A.1 for details).

of our model, i.e., the incremental refinement of internal representation by interative integration of top-down and bottom-up information. Our net's internal representation is incremetally and iteratively refined over time by integrating the top-down contextual information conveyed through the feedback recurrent connnections and the higer spatial-frequency information relayed through the bottom-up feedforward connections.

Note that we did not use weight sharing in the neuronal boundary detection net, whereas we used weight sharing between some convolutions at the same height in the error-detecting and error-correcting nets (Figure 2 and Figure S12).

Architectural details 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 (Figure S10). The receptive field of a neuron in the higher layers is therefore nearly isotropic and roughly cubic in physical size. To limit the number of parameters in our model, we factorized all 3D convolutions into a 2D convolution followed by a 1D convolution in z-dimension. We employed exponential linear units (ELUs, [6]) as nonlinearity, except for the output layer with logistic activation functions. We trained our nets to generate a 3D affinity graph [38, 21, 9], thus outputting three channels corresponding to x, y, and z-affinity map, respectively.

A.2 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.

Human experts produced multiple volumes of gold standard dense reconstruction, in total 20 volumes of size $512 \times 512 \times 100$. We trained our boundary detector using 19 volumes and used the last volume for training validation. We applied the trained boundary detector on a new image volume of size $2048 \times 2048 \times 100$ to obtain a preliminary segmentation, which was then proofread by the experts to generate a bootstrapped ground truth volume. This volume was used to optimize the parameters for watershed and mean affinity agglomeration [21]. Finally, the optimized segmentation pipeline was applied to generate further bootstrapped ground truth for the error detection and correction tasks.

A.3 Training procedures

Our boundary detection networks were implemented in the Caffe deep learning framework [18]. To train our nets, we minimized the pixelwise binomial cross-entropy loss with class-rebalancing using the Adam optimizer [20], initialized with $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 0.01$. The network weights were initialized following He et al. [11]. The learning rate (or step size parameter α in the Adam optimizer) was halved when validation loss plateaued out, five times in total at 35K, 175K, 250K, 300K, and 480K training iterations. We used a single patch of size $158 \times 158 \times 32$ (i.e. minibatch of size 1) to compute gradients at each training iteration. Each training sample was augmented with rotations, reflections, warping, brightness and contrast perturbations, and simulated misalignments [21]. The training lasted for 800K iterations until convergence, which took about five days on a single NVIDIA Titan X Pascal GPU.

A.4 Inference and postprocessing

We performed *overlap-blending* inference followed by watershed and mean affinity agglomeration [21]. We refer the interested readers to [21] for further details.

B Per-object VI score

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

$$\begin{aligned}
VI_{\text{split}} &= -\frac{1}{\sum_{i,j} r_{ij}} \sum_{i,j} r_{ij} \log\left(r_{ij}/p_i\right), \\
VI_{\text{merge}} &= -\frac{1}{\sum_{i,j} r_{ij}} \sum_{i,j} r_{ij} \log\left(r_{ij}/q_j\right), \\
p_i &= \sum_j r_{ij}, \\
q_j &= \sum_i r_{ij},
\end{aligned}$$

where r_{ij} is the number of voxels in common between the i^{th} segment of the ground truth segmentation and the j^{th} segment of the proposed segmentation [27].

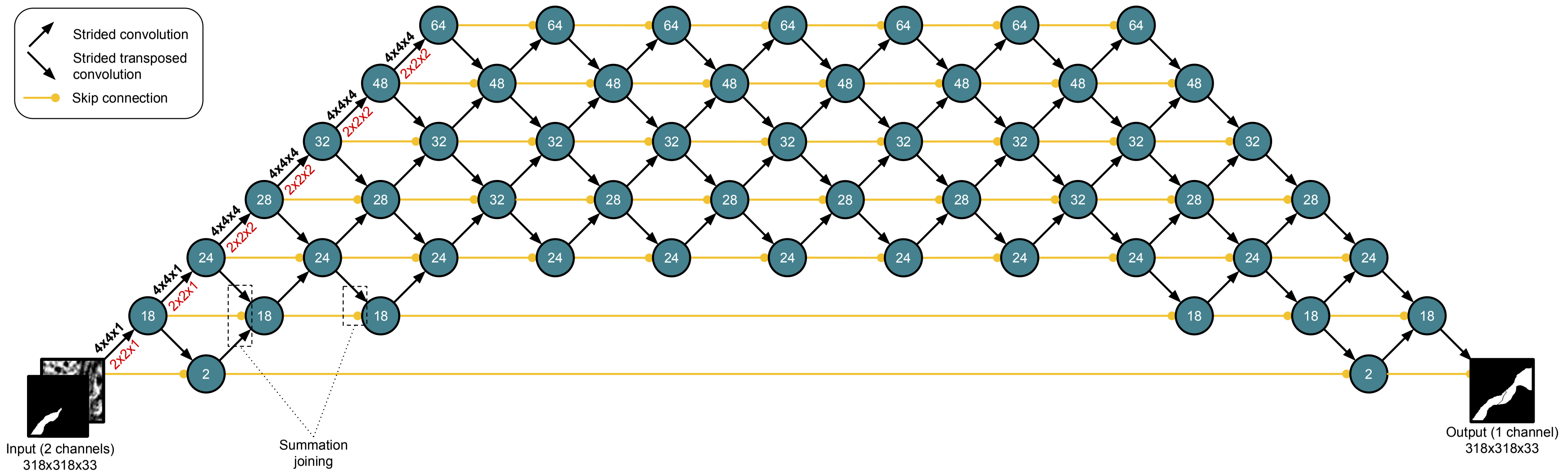
We define the split and merge scores for ground truth segment i as

$$\begin{aligned}
VI_{\text{split}}(i) &= -\sum_j r_{ij}/p_i \log(r_{ij}/p_i), \\
VI_{\text{merge}}(i) &= -\sum_j r_{ij}/p_i \log(r_{ij}/q_j).
\end{aligned}$$

Both quantities have units of nats. $VI_{\text{split}}(i)$ is zero if and only if ground truth segment i is contained within a segment in the proposed segmentation, while $VI_{\text{merge}}(i)$ is zero if and only if 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 $VI_{\text{split}}(i)$, $VI_{\text{merge}}(i)$ respectively.

C Training details

The error-detecting and error-correcting networks were implemented in TensorFlow [1] and trained using 4 TitanX Pascal GPUs with synchronous gradient descent. We used the Adam optimizer, initialized with $\alpha = 0.001$, $\beta_1 = 0.95$, $\beta_2 = 0.9995$, and $\epsilon = 0.1$ [20]. Both nets were trained until the loss on a validation set plateaued. The error-detecting net was trained for 700K iterations (approximately one week), while the error-correcting net was trained for 1.7M iterations (approximately three weeks).



Strided convolution
Strided transposed convolution
Skip connection
Input (2 channels) 318x318x33
Output (1 channel) 318x318x33
Summation joining
4x4x1
2x2x1
4x4x4
2x2x2
64
48
32
28
24
18
2

Figure S12: Error-correcting network architecture.