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

Significant advances have been made in recent years on the problem of neural circuit reconstruction from electron microscopic imagery. Improvements in image acquisition, image alignment, and boundary detection have greatly reduced the achievable error rate. In order to make further progress, we argue that automated error detection is essential for focusing the effort and attention of both human and machine. In this paper, we report on the use of automated error detection as an attention signal for a flood filling error correction module. We demonstrate significant improvements upon the state of the art in segmentation performance.

1 Introduction

A promising approach to understanding the fine structure of neural circuitry is the use of serial sectioning combined with electron microscopy to produce nanoscale resolution volumetric imagery of neural tissue. In order to convert such images into a wiring diagram, the neurites in a given volume of tissue must be segmented.

State-of-the-art segmentation pipelines for this problem use convolutional neural networks to detect boundaries between cells, followed by a connected components/watershed transform and hierarchical agglomeration procedures [18, 7]. In order to obtain a completely correct segmentation suitable for science, human tracers manually proofread the resulting segmentation. With the declining error rate of automated segmentation, a growing proportion of human time is spent searching for errors in a segmentation. Futhermore, we observe the interesting phenomenon that it is often much easier to detect an error than to find the correct segmentation. Indeed, humans are usually able to detect a segmentation error without even looking at the original image; they look for neurites that terminate prematurely or x-shaped junctions indicating incorrectly merged segments. However, the incredible density of information in neural tissue makes searching for the correction difficult.

Our approach is to decompose the problem of refining a given segmentation into two easier problems: error detection and error correction. As a wise man once said, recognizing your faults is the first step towards fixing them. We hypothesize that thanks to the distinctive shapes of neurons, recognizing an error is much easier than finding the correct answer.

Conversely, an error detection module makes error correction much easier. Once we remove from consideration all objects which do not contain an error, the visual crowding problem is alleviated. Choosing the correct extension of a neurite is easy when there are only a few choices available. Furthermore, we can afford to apply a relatively expensive error correction procedure as it does not need to be applied everywhere.

We implement both error detection and error correction using 3d multiscale convolutional neural networks.

In this paper, we will demonstrate the feasibility of high quality error detection. We will also demonstrate the effectiveness of error detection in directing the attention of an error correction

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

module. We expect our error detector will be independently useful for directing human proofreading effort.

2 Related Work

Within the context of the neuron segmentation problem, the most directly comparable technique is work on learned policies for supervoxel agglomeration [4, 13]. Whereas the primitive operation in supervoxel agglomeration is a black box which accepts a pair of segments and decides whether to merge them, our primitive operation accepts a single segment and decides whether it should be modified in any way. In one shot, our error correction network then selects from among many choices the correct change to make. Both methods consider only one or two objects at a time and therefore benefit from focused attention and shape cues. However, greedy agglomeration must evaluate each pair of touching objects at least once, while our pipeline need only evaluate most objects once. The additional overhead in agglomeration prohibits the use of expensive deep learning. Some progress in this direction has been made (see [2]), but the standard approach remains the use of hand designed features [13].

We also argue that the error detection task is better posed than the supervoxel agglomeration task. Given two objects which already contain errors, it is often unclear whether the segmentation improves after they have been merged. In [4], the authors resolve this ambiguity by training to predict the change in rand score from a merge.

Recent efforts at error detection have been made in [9] (which uses topological features of a skeletal representation) and [15] (which uses convolutional neural networks).

Our error correction approach is closely related to the work on flood-filling networks in [5] and [9], which in turn is related to [16] and [14]. As in their approach, we train a neural network to reconstruct one object at a time. Our main novelty relative to their approach is the use of the advice of an error detector to bias flood filling. While they present their network with a partially reconstructed object and ask for a completion, we present our network with the union of all possibly incorrect segments in a window and ask the network to split out a single object. Since the typical error rate is already low, this "advice" on which objects to consider is informative and significantly improves the performance of our flood filling networks. While they perform inference densely, we selectively apply our flood filling networks near likely errors. This comparatively reduces our computational cost. We sacrifice end-to-end training for these advantages.

Within the broader context of machine learning, our approach may be compared to other strategies for structured prediction problems. A relevant approach is the use of the conditional generative adversarial framework, in which one simultaneously trains a prediction network and an discriminator network which enforces structural constraints on the output [10, 8]. We do not co-train our error correction and error detection networks, but this could be the subject of future work.

Our approach may also be compared with models for visual attention in the literature (for example, [11]). Recurrent neural networks are able to learn in an end-to-end way how to find which parts of an image are relevant to the given task. In contrast, we have a fixed policy for which objects to attend to: we attend to those objects which likely contain errors. One of our central findings is that this policy is highly selective and improves segmentation performance.

3 Error Detection

3.1 Task Specification

We take a fully supervised approach to error detection. Given a window of size $p_x \times p_y \times p_z$ and a binary mask encoding a single object in a proposed segmentation, the error detection task is to report whether or not the restriction of the mask to the window is pixel-wise equal to the restriction of some object in the ground truth. Our use of pixel-wise equality demands at training-time a ground-truth segmentation that is composed of the same building blocks (supervoxels) as the proposed segmentation.

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, it is possible that two objects participating in a

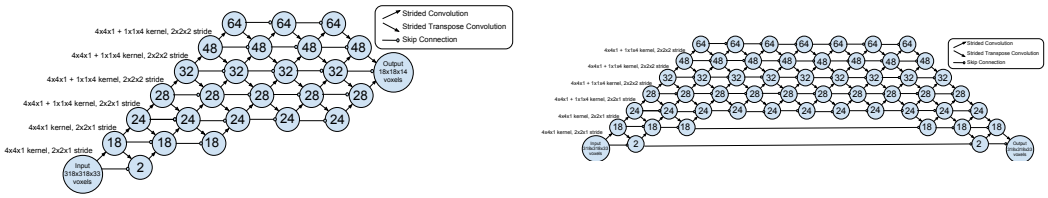

Figure 1: Architectures for the error detection and error correction modules respectively. Each node represents a layer and the number inside represents the number of feature maps. The layers closer to the top of the diagram have lower resolution than the layers near the bottom. We make savings in computation by minimizing the number of high resolution feature maps. The diagonal arrows represent strided convolutions, while the horizontal arrows represent skip connections.

merge error never appear in the same window. These merge errors would not be classified as an error in any window. For our experiments, we chose a safe size of $p_x \times p_y \times p_z = 46 \times 46 \times 7$.

3.2 Architecture

We implement error detection using a multiscale 3d convolutional neural network. The architecture is detailed in figure 1. Its design is informed by experience with convolutional networks for boundary detection (see [7]) and reflects recent trends in neural network design [17, 3]. Its input patch size 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). We trained two variants, one of which takes as input only the object mask, and another which additionally receives as input the original image. The network simultaneously reports errors in several overlapping subwindows of size $p_x \times p_y \times p_z$ arranged in an $18 \times 18 \times 14$ grid.

4 Error Correction

4.1 Task Specification

Our error correction module takes as input an image patch of size (P_x, P_y, P_z). It takes an auxiliary "advice" input of the same size which is a binary mask in which some objects have been zeroed out. At test time, objects which contain no detected errors within the given window will be zeroed out. The desired output is a binary image encoding the object which overlaps the central pixel. (The central object is guaranteed not to be masked out).

4.2 Architecture

Yet again, we implement error correction using a multiscale 3d convolutional neural network. The architecture is detailed in figure 1. 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 output 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:

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

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. When we move the output window, the network may output the same vector field v.

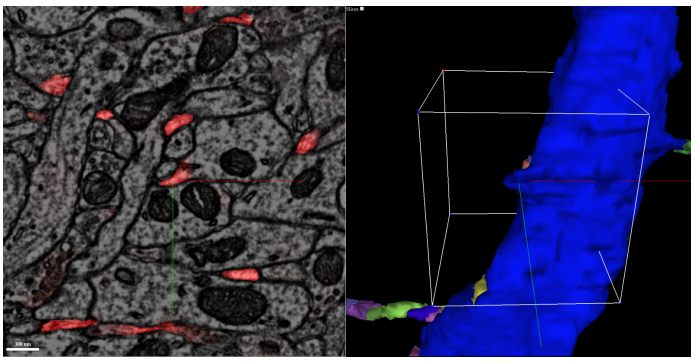

Figure 2: An example of a mistake in the initial segmentation. The dendrite is missing a spine. The red overlay on the left shows the output of the error detector; the stump in the centre of the image was clearly marked as an error.

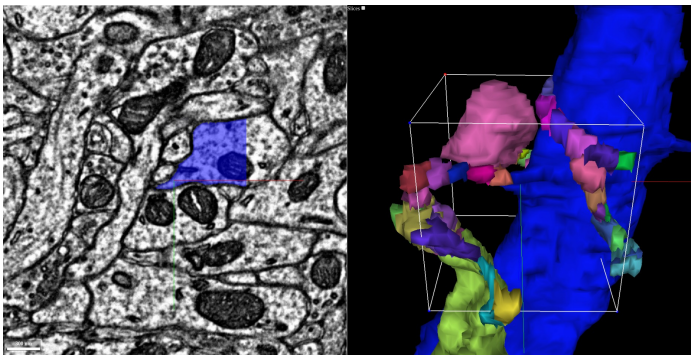

Figure 3: The right shows all objects which contained a detected error in the vicinity. This is the "advice" image which is provided as an auxiliary input to the flood filling network. For clarity, these objects were clipped to lie within the white box representing the field of view of our flood filling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

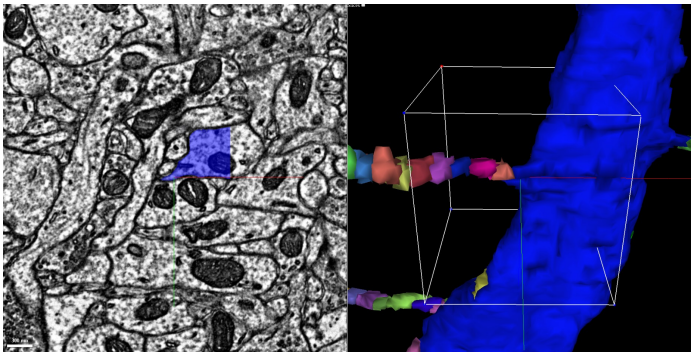

Figure 4: The supervoxels assembled in accordance with the output of the flood filling network.

5 Putting it Together

In this section, we describe how error detectors and error correctors are combined at test time. We begin with a proposed segmentation whose remaining errors are assumed to be sparsely distributed. We first densely sample points in the segmentation. We run the error detector at sample locations until every point in every object is covered by an output window at least twice. When the error detector disagrees with itself, we conservatively take the maximum output.

During the error correction phase, we maintain a graph whose vertices are segments in an over-segmentation (henceforth called supervoxels) and whose connected components are segments in the proposed segmentation. In a window centred on each detected error, we produce a binary mask with all supposedly-error-free objects zeroed out. We then apply the error correction network which produces a binary image M representing the object containing the central pixel. The intensities in M are averaged within each supervoxel. If the output passes a confidence threshold, we add to the regiongraph a clique on those supervoxels with high average intensity in M and delete all edges connecting these supervoxels to others in the window. The error detection network is then reapplied locally. We iterate until every location is either error free or has been corrected at least twice.

6 Experiments

6.1 Dataset

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

A team of tracers produced a gold standard dense reconstruction of 850 Mvoxels. This volume was used to train the boundary detection networks. We applied the resulting boundary detector to a larger volume of size 5700 Mvoxels. Tracers corrected the resulting segmentation. This bootstrapped ground truth was used to train the error detector and error corrector. A subvolume of size 910 Mvoxels was reserved for validation, and two volumes of size 910 Mvoxels were reserved for testing.

Producing the gold standard segmentation required a total of ~ 900 tracer hours, while producing the bootstrapped ground truth required ~ 670 tracer hours.

6.2 Baseline Segmentation

Our baseline segmentation is produced using a standard pipeline of multiscale convolutional neural networks for boundary detection, watershed, and mean affinity agglomeration. A similar pipeline is described in detail in [7]. The segmentation performance values reported for the baseline are taken at a mean affinity agglomeration threshold of 0.23, which minimizes the total variation of information error metric on the test volume.

6.3 Training

The neural networks were implemented in TensorFlow [1] and trained using 4 TitanX Pascal GPUs with synchronous gradient descent. We used the Adam optimizer [6]. We augmented all of our data with rotations, simulated misalignments, and missing sections. The proposed segmentations used for training the error detection network were generated using mean affinity agglomeration at a threshold of 0.3. 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).

6.4 Error Detection Results

To measure the quality of error detection, we densely sampled points in our test volume. In order to remove ambiguity over the precise location of errors, we sampled only points which contained an error within a surrounding window of size $40 \times 40 \times 4$ or did not contain an error within a surronding window of size $80 \times 80 \times 8$. 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.

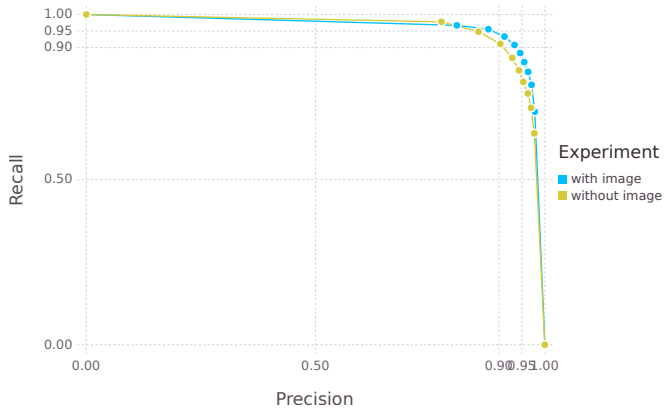

Figure 5: Precision and recall for error detection, both with and without access to the original image. In the test volume, there are 8266 error free locations and 961 locations with errors. In practice, we use threshold which guarantees > 95% recall and > 85% precision.

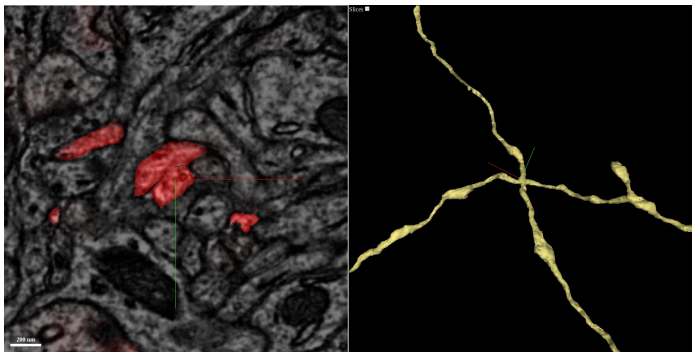

Figure 6: An example of a detected error. The right shows two incorrectly merged axons, and the left shows the output of the error detector overlaid on the corresponding 2d image in red.

We trained one error detector with access to the original 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.

6.5 Error Correction Results

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**

In order to demonstrate the importance of error detection in error correction, we applied our error correction algorithm both with and without the auxiliary "advice" input channel having error-free objects zeroed out. The error correction network was simultaneously trained with and without advice, so this comparison is fair. 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 variation of information or rand score since changes can be dominated by modifications to a few large objects. Therefore, we

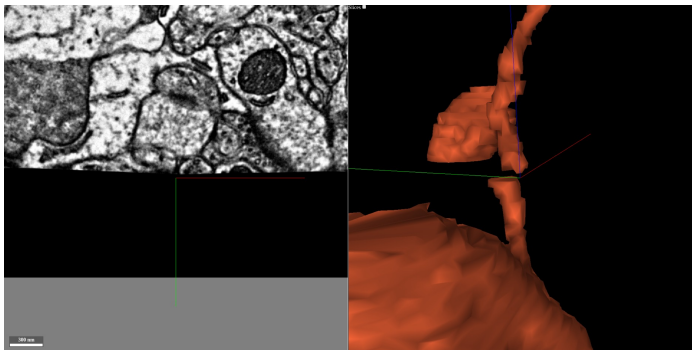

Figure 7: A difficult location with missing data in one section combined with a misalignment between slices. Our algorithm was able to correctly merge across the missing data.

decomposed the variation of information into a score for each object in the ground truth. 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 [12].

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 $VI_{split,merge}$ is a weighted sum of the per-object scores $VI_{split,merge}(i)$. 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.

Given that our baseline approach already produces state-of-the-art results on other datasets (see [7]), we expect that the method presented here is a substantial improvement upon the state of the art. However, we have not conducted experiments on publicly available datasets, and therefore we leave a careful comparison for future work.

6.6 Cost Analysis

Table 2 shows the cost of the most expensive parts of our segmentation pipeline. The combined cost of error detection and error correction is within an order of magnitude of the cost of 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 segmentation decreases and the precision of the error detector increases, the number of locations requiring error correction will only fall further.

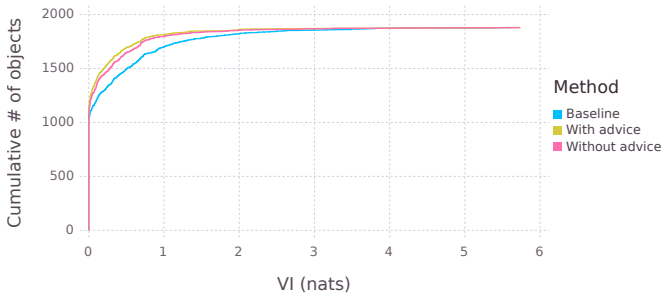

Figure 8: Per-object vi scores for the 940 reconstructed objects in our test volume. Almost 800 objects are completely error free in our segmentation. These objects are likely all axons; almost every dendrite has a couple of errors.

Table 2: Computation time for a volume of size $2048 \times 2048 \times 256$ using a single TitanX Pascal GPU

Boundary Detection	20 mins
Error Detection	25 mins
Error Correction	55 mins

7 Conclusion and Future Directions

We have developed a segmentation error detector and demonstrated its efficacy in biasing the attention of flood filling networks. 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 original 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 flood filling networks can benefit from the advice of error detection, improving segmentation performance upon our state-of-the-art 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.

8 Author Contributions

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.

9 Acknowledgements

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] John A. Bogovic, Gary B. Huang, and Viren Jain. Learned versus hand-designed feature representations for 3d agglomeration. *CoRR*, abs/1312.6159, 2013. URL `http://arxiv.org/abs/1312.6159`.

[3] 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`.

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

[5] 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`.

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

[7] 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`.

[8] 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`.

[9] 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.

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

[11] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. Recurrent models of visual attention. *CoRR*, abs/1406.6247, 2014. URL `http://arxiv.org/abs/1406.6247`.

[12] 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`.

[13] 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/`.

[14] 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.

[15] 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.

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

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

[18] 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.

A Network Specifications

The layers along each horizontal sequence in the 1 have the same size. Their sizes are shown detailed in 3. 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 is z. We also use weight sharing between some convolutions at the same height.

Table 3: Error Detector Convolution Kernel Sizes

Height	Layer Size	Convolutional Kernel Size	Convolutional Kernel Stride
1	$2 \times 318 \times 318 \times 33$	$4 \times 4 \times 1$	$2 \times 2 \times 1$
2	$4 \times 158 \times 158 \times 33$	$4 \times 4 \times 1$	$2 \times 2 \times 1$
3	$24 \times 78 \times 78 \times 33$	$4 \times 4 \times 4$	$2 \times 2 \times 1$
4	$28 \times 38 \times 38 \times 30$	$4 \times 4 \times 4$	$2 \times 2 \times 2$
5	$32 \times 18 \times 18 \times 14$	$4 \times 4 \times 4$	$2 \times 2 \times 2$
6	$48 \times 8 \times 8 \times 6$	$4 \times 4 \times 4$	$2 \times 2 \times 2$
7	$64 \times 3 \times 3 \times 2$	$4 \times 4 \times 4$	$2 \times 2 \times 2$

Table 4: Error Corrector Convolution Kernel Sizes

Height	Layer Size	Convolutional Kernel Size	Convolutional Kernel Stride
1	$2 \times 318 \times 318 \times 33$	$4 \times 4 \times 1$	$2 \times 2 \times 1$
2	$18 \times 158 \times 158 \times 33$	$4 \times 4 \times 1$	$2 \times 2 \times 1$
3	$24 \times 78 \times 78 \times 33$	$4 \times 4 \times 4$	$2 \times 2 \times 1$
4	$28 \times 38 \times 38 \times 30$	$4 \times 4 \times 4$	$2 \times 2 \times 2$
5	$32 \times 18 \times 18 \times 14$	$4 \times 4 \times 4$	$2 \times 2 \times 2$
6	$48 \times 8 \times 8 \times 6$	$4 \times 4 \times 4$	$2 \times 2 \times 2$
7	$64 \times 3 \times 3 \times 2$	$4 \times 4 \times 4$	$2 \times 2 \times 2$