Convolutional Networks Can Learn to Generate Affinity Graphs for Image Segmentation

LETTER **Communicated by Dmitri Chklovskii**

Convolutional Networks Can Learn to Generate Affinity Graphs for Image Segmentation

Srinivas C. Turaga*
sturaga@mit.edu
Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology, Cambridge, MA 02139, U.S.A.

Joseph F. Murray*
jfmurray@jfmurray.org
Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology, Howard Hughes Medical Institute, Cambridge, MA 02139, U.S.A.

Viren Jain
viren@mit.edu
Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology, Cambridge, MA 02139, U.S.A.

Fabian Roth
fabian.roth@gmail.com
Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology, Howard Hughes Medical Institute, Cambridge, MA 02139, U.S.A.

Moritz Helmstaedter
Moritz.Helmstaedter@mpimf-heidelberg.mpg.de
Kevin Briggman
Kevin.Briggman@mpimf-heidelberg.mpg.de
Winfried Denk
Winfried.Denk@mpimf-heidelberg.mpg.de
Max-Planck Institute for Medical Research, D-69120, Heidelberg, Germany

H. Sebastian Seung
seung@mit.edu
Departments of Brain and Cognitive Sciences and Physics, Massachusetts Institute of Technology, Howard Hughes Medical Institute, Cambridge, MA 02139, U.S.A.

Many image segmentation algorithms first generate an affinity graph and then partition it. We present a machine learning approach to computing

*These authors contributed equally to the writing of this letter.

an affinity graph using a convolutional network (CN) trained using ground truth provided by human experts. The CN affinity graph can be paired with any standard partitioning algorithm and improves segmentation accuracy significantly compared to standard hand-designed affinity functions.

We apply our algorithm to the challenging 3D segmentation problem of reconstructing neuronal processes from volumetric electron microscopy (EM) and show that we are able to learn a good affinity graph directly from the raw EM images. Further, we show that our affinity graph improves the segmentation accuracy of both simple and sophisticated graph partitioning algorithms.

In contrast to previous work, we do not rely on prior knowledge in the form of hand-designed image features or image preprocessing. Thus, we expect our algorithm to generalize effectively to arbitrary image types.

1 Introduction

Image segmentation algorithms aim to partition pixels into domains corresponding to different objects. Graph-based algorithms solve the segmentation problem by constructing and then partitioning an affinity graph where the nodes correspond to image pixels or image regions. Edges between image pixels are weighted and reflect the affinity between nodes. Affinity functions used to compute the edge weights are traditionally hand-designed and use local image features such as image intensity, spatial derivatives, texture, or color to estimate the degree to which nodes correspond to the same segment (Shi & Malik, 2000; Fowlkes, Martin, & Malik, 2003). In this letter, we have pursued a different approach, which is to use learning to generate the affinity graph. Using a machine learning architecture known as a convolutional network (LeCun et al., 1989) and ground truth segmentations generated by human experts, we train a function that maps a raw image directly to an affinity graph. This learned affinity graph can then be partitioned using any standard partitioning algorithm.

It has been recognized for some time that the performance of a graph-based segmentation algorithm can be hampered by poor choices in affinity function design. To this end, there has been a limited amount of prior work on learning affinity functions from training data sets. Much prior work has been confined to estimating affinities by classifying carefully chosen hand-designed image features, which are often image domain specific (Fowlkes et al., 2003). This may have been for lack of a good image processing architecture that can discover the appropriate features directly from the raw image. Here, we present a new approach using convolutional networks. With a rich architecture possessing many thousands of free parameters, it can in principle extract and effectively utilize a huge variety of image features. We demonstrate in practice that a gradient descent procedure is able

to adjust the many parameters of the network to achieve good segmentation performance. Since our approach does not contain image-domain-specific assumptions, we expect it to generalize to images from diverse application.

We have conducted quantitative performance tests on a database of electron microscopic brain images described below. The tests demonstrate a marked improvement in the quality of segmentations resulting from partitioning our learned affinity graph versus commonly used heuristic affinity functions.

We consider the segmentation of neural processes, such as axons, dendrites, and glia, from volumetric images of brain tissue taken using serial block-face scanning electron microscopy (SBF-SEM; Denk & Horstmann, 2004). This technique produces 3D images with a spatial resolution of roughly 30 nm or better (see Figure 1A). If applied to a tissue sample of just 0.5 mm $\times$ 0.5 mm $\times$ 1.2 mm from the cerebral cortex, the method would yield a 3D image of several tens of teravoxels. Such a volume would contain about 15,000 neurons and many thousands of nonneuronal cells (Smith, 2007). Neurons are particularly challenging to segment due to their highly branched, intertwined, and densely packed structure. In addition, at the imaging resolution, axons can be as narrow as just a few voxels wide (Briggman & Denk, 2006). The sheer volume and complexity of data to be segmented preclude manual reconstruction and make automated reconstruction algorithms essential. Small differences in segmentation performance could lead to differences in weeks to months of human effort to proofread machine segmentations.

Prior work on electron and optical microscopic reconstruction of neurons has ranged the spectrum from completely manual tracing (White, Southgate, Thomson, & Brenner, 1986; Fiala, 2005) to semiautomated interactive methods (Liang, McInerney, & Terzopoulos, 2006; Carlbom, Terzopoulos, & Harris, 1994) and fully automated methods (Mishchenko, 2009; Helmstaedter, Briggman, & Denk, 2007; Jurrus et al., 2009; Andres, Köthe, Helmstaedter, Denk, & Hamprecht, 2008). White et al. (1986) took over 10 years to complete the manual reconstruction of the entire nervous system of a nematode worm *C. elegans*, which contains only 302 neurons, underscoring the need for significant automation. Most attempts at automation have involved hand-designed filtering architectures with little machine learning (Vasilevskiy & Siddiqi, 2002; Al-Kofahi et al., 2002; Jurrus et al., 2009; Mishchenko, 2009; Andres et al., 2008). For example, Jurrus et al. (2009) first do an extensive denoising procedure, followed by an initial 2D segmentation and then a Kalman filter to track an object outline through successive sections. Carlbom et al. (1994) adapted the widely used active contour (snakes) technique for semiautomated neuron tracing. However, this process is interactive and requires human selection of seed points and careful controlling of parameters for each neuron to be segmented. In contrast to these approaches, Helmstaedter et al. (2007) use supervised learning in the form of a nearest-neighbor classifier to segment expectation-maximization

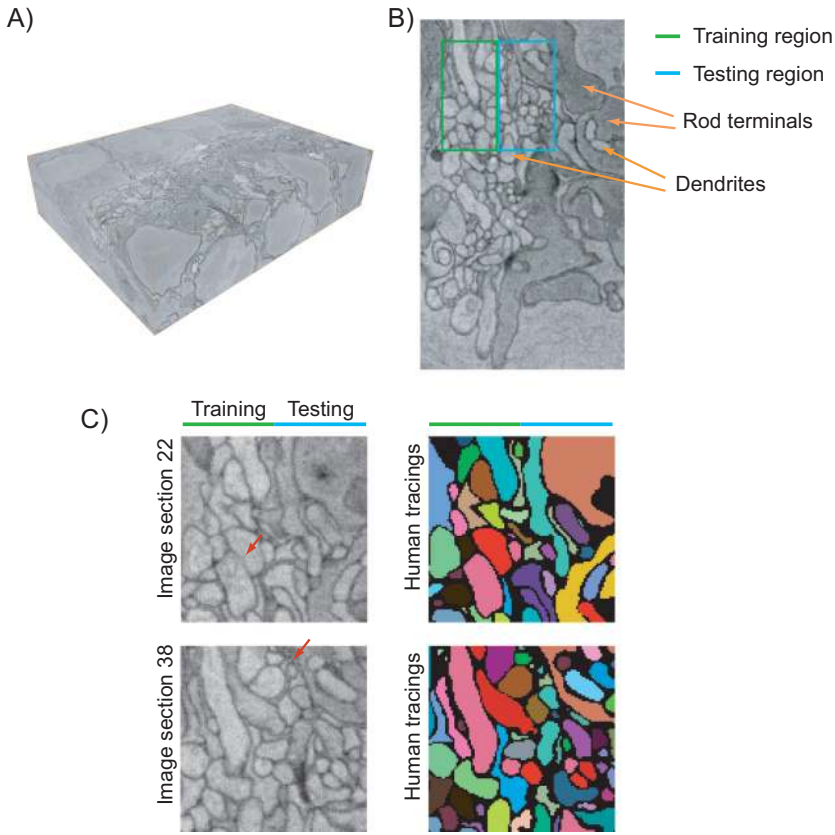

Figure 1: (A) View of 3D stack of images generated by serial block-face scanning electron microscopy (SBF-SEM). Tissue is from the outer plexiform layer of the rabbit retina. The total volume is 800 × 600 × 100 voxels corresponding to 20.96 × 15.72 × 5 μm^3. (B) Larger view of a section from this volume. The region traced by humans is divided into training and test sets, shown in green and blue. Some of the large gray objects are axon terminals of light-sensitive rod cells, while many of the smaller, brighter objects are dendrites. (C) Original images and human tracings of two sections, each 100 × 100 × 100 voxels, where each color indicates a different process. Red arrows indicate challenging regions for segmentation algorithms. In section 22 (top), the boundary between two processes is faint, yet these processes are segmented as different objects. In section 38 (bottom), some processes become very small, less than 10 voxels in area through this section.

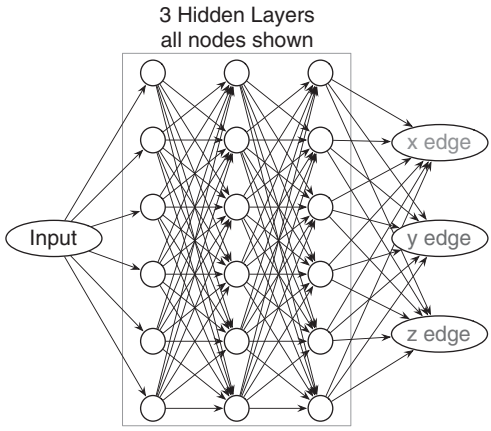

Figure 2: The convolutional network architecture used for our experiments contains four layers of convolutions with six feature maps each and three output images. Each node is an image, and each edge represents convolution by a filter.

(EM) images, eliminating the need for human interaction and parameter tuning. Other popular segmentation methods make use of Markov random fields (MRFs). Convolutional networks are closely related to but provide superior performance to MRFs, as explained in our prior work (Jain et al., 2007).

The paper is organized as follows. We first define convolutional networks in section 2. Section 3 describes the structure of the affinity graph and how we apply convolutional networks to produce this graph directly from the raw image. In section 4 we present results from various network architectures and graph partitioning algorithms and quantitatively evaluate the results against human labeled ground truth. We conclude in section 5 with some future directions, and the three appendixes give algorithmic details and performance metrics.

2 Convolutional Networks Are a Powerful Image Processing Architecture

Neural networks (NN) have been long used for image processing applications (Egmont-Petersen, de Ridder, & Handels, 2002; Sinthanayothin, Boyce, Cook, & Williamson, 1999; Nekovei & Sun, 1995). Although convolutional networks (CNs) are closely related to NNs, here we simply define CNs without explaining the relationship. A detailed explanation can be found in appendix A.

Formally, a CN is defined on a directed graph as in Figure 2. (The directed graph representing a CN must not be confused with the undirected

affinity graph.) The ath node of the graph corresponds to an image-valued activation I_a and a scalar-valued bias h_a, and the edge from node b to a corresponds to the convolution filter w_{ab}. In addition, there is a smooth activation function, often the sigmoid $f(x) = (1 + e^{-x})^{-1}$. The images at different nodes are related by

$$I_a = f\left(\sum_b w_{ab} * I_b + h_a\right). \tag{2.1}$$

The sum runs over all nodes b with edges into node a. Here $w_{ab} * I_b$ represents the convolution of image I_b with the filter w_{ab}. After adding the bias h_a to all pixels of the summed image, I_a is generated by applying the element-wise nonlinearity $f(x)$ to the result.

The CNs studied in this letter are based on graphs with nodes grouped into multiple layers. In an image processing application, the first layer typically consists of a single input image. The final layer consists of one or more output images. In our case, we shall generate several output images—one image for each edge class in the nearest-neighbor graph (see Figure 3A). The intermediate or hidden layers also consist of images, or feature maps, which represent the features detected, and are the intermediate results of the overall computation.

The filters w_{ab} and the biases h_a constitute the adjustable parameters of the CN. Features in the image are detected by convolution with the filters w_{ab}. Thus, learning the filters from the training data corresponds to automatically learning the features that are most useful to the classifier. The role of the values h_a is to bias the network for or against the detection of the features. The training procedure for h_a can be thought of as finding the correct a priori belief as to the existence of certain features in the image.

The well-known error backpropagation algorithm for training NNs can be easily generalized for training w_{ab} and h_a in a CN (LeCun, Bottou, Bengio, & Haffner, 1998; LeCun, Bottou, Orr, & Muller, 1998). This gradient descent procedure minimizes a cost function that measures the discrepancy between the output image produced by the CN, I_O, and a desired output image I_O^d, $\mathcal{L}(I_O, I_O^d) = \sum_{pixels} \frac{1}{2}(I_O - I_O^d)^2$. The gradient of the CN parameters with respect to this cost function can be computed using the following equations:

$$\begin{aligned} S_O &= \left(I_O - I_O^d\right) \odot f'(I_O) \\ S_b &= \left(\sum_a S_a \circledast w_{ab}\right) \odot f'(I_b) \\ \Delta w_{ab} &= \eta I_a \circledast S_b \\ \Delta h_a &= \eta \sum_{pixels} S_a. \end{aligned} \tag{2.2}$$

These equations implement an efficient recursive gradient computation that gives the backpropagation algorithm its name. While equation (2.1) implements a recursive computation where information flows in the direction indicated by the directed edges of the CN graph, the recursion in equation 2.2 progresses in the opposite direction. The sum in this equation is over all edges leading out of a given node b. Here $S_a \circledast w_{ab}$ represents the cross-correlation of the image S_a with the filter w_{ab}, $\odot$ a pixel-wise image multiplication operation and $f'(x)$ the derivative of the nonlinearity $f(x)$. The size of the gradient update at each iteration is controlled by the constant η.

In the past, convolutional networks have been successfully used for image processing applications such as object recognition, handwritten digit classification, and cell segmentation (LeCun et al., 1989; Ning et al., 2005). For recognition tasks, the network is trained to produce a categorical classification, such as the identity of the digit. In the cell segmentation application (Ning et al., 2005), a CN produces five outputs (such as cell wall, nucleus, outside medium) for every input pixel. In our case, we use these networks to perform a nonlinear transformation that maps one 3D image to a set of other 3D images. There are two important contrasts between our work and that of Ning et al. (2005). First, our architecture does not include subsampling layers, which means that all filters at every layer are learned, rather than some layers containing predefined down-sampling filters. This leads to output images that are at the same resolution as the input image. Second, we use these networks to generate a nearest-neighbor graph, where the number of edges is proportional to the size of the image. This allows our networks (in combination with a graph partitioning algorithm) to segment objects that may have no boundary pixels separating them (in contrast to Ning et al., 2005, which would require at least one pixel separation between adjacent objects). This is important in our application, where we are severely resolution limited when compared to the thickness of cell boundaries and neurites.

3 Using Convolutional Networks to Generate an Affinity Graph

Figure 3 illustrates the computational problem solved in this letter using convolutional networks. We wish to map an input image to a set of affinities corresponding to the edges of an affinity graph. The nodes of the graph represent image voxels, which form a three-dimensional cubic lattice. There are edges between all pairs of nearest neighbors. Each node is connected to six neighbors in the x, y, and z directions, as shown in the right side of Figure 3A. Since each edge is shared by a pair of nodes, there are three times as many edges as nodes. As shown in Figure 3A, we can think of the affinity graph as three different images, each representing the affinities of the edges of a particular direction. Therefore, the problem can be seen as

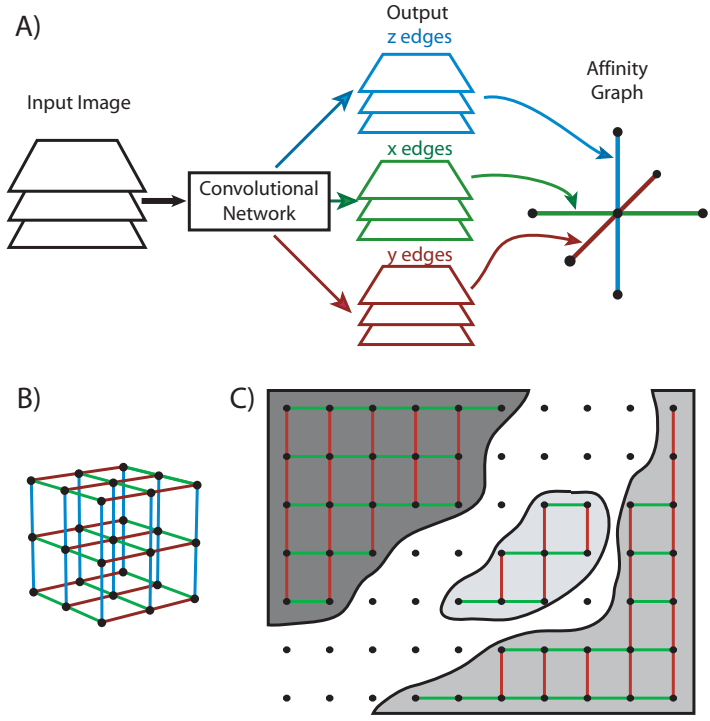

Figure 3: (A) Creating the affinity graph using a convolutional network. The input to the network is the 3d EM image and the desired output is a set of 3D images: one each for the x, y, and z directions representing the affinity graph. (B) The edges of the nearest-neighbor affinity graph form a lattice. (C) Desired affinities for a set of three segments (gray). Edges contained within a segment have desired affinity 1 (green for x edges and red for y edges). Edges not contained within a segment have desired affinity 0, implying that boundary voxels are disconnected from themselves.

the transform of one input image into three output images representing the affinity graph.

Ultimately our goal is to generate an image segmentation. This is accomplished by using a graph-partitioning algorithm to cut the affinity graph into a set of discrete objects. This partitioning segments the image by cutting weak affinity edges to create clusters of nodes corresponding to different image segments. The computational problem of segmentation thus involves two steps: the transformation of the input image into an affinity graph and then the partitioning of this graph into a segmentation. The quality of a segmentation can be improved by generating affinity graphs that accurately indicate segment boundaries.

We define the desired affinity graph as a same-segment indicator function. Edges between nodes in the same segment have value 1, and edges between nodes not contained in the same segment have value 0 (see Figure 3C). Further, boundary voxels are also defined to have 0 affinity with each other. This causes boundary voxels to form separate segments; this is a subtlety in our definitions that leads to a more robust segmentation. If reproduced faithfully, this affinity graph should lead to perfect segmentation, since the graph can be easily partitioned along segment boundaries. It should be noted, however, that this is only one of many affinity graphs that are capable of yielding perfect segmentation.

4 Results

In this section, we first demonstrate the ability of convolutional networks to produce affinity graphs directly from the images. We find that after training, our CN is able to correctly predict the affinity between two voxels about 90% of the time. This is comparable to the level of agreement between two humans concerning object boundaries. We then test the segmentation performance of this affinity graph using two graph-partitioning algorithms: the well-known normalized cuts and a simpler connected-components method. The segmentation using our affinity graph is dramatically better than with a standard affinity graph; this is quantified using three different segmentation metrics and compared against interhuman variability in segmentation by experts. We also find that our improved affinity enables partitioning by connected components to be more accurate at segmentation than the normalized cuts algorithm. This is a pleasant surprise, as connected components also run much faster than normalized cuts.

4.1 About the Data Set. We use images of retinal tissue collected with the serial block-face scanning electron microscope (SBF-SEM; Denk & Horstmann, 2004). We imaged a volume of dimension $21 \times 15.6 \times 5\ \mu\text{m}^3$, corresponding to $800 \times 600 \times 100$ voxels at a resolution of $26 \times 26 \times 50\ \text{nm}^3$ (see Figure 1A). Cell bodies compose the majority of the volume at the sides, while much smaller neurites are more common in the center. To focus on neurites, a central region of $320 \times 200 \times 100$ voxels was selected for investigating automated segmentations (see Figure 1B). A subvolume of $100 \times 100 \times 100$ voxels was completely segmented by two human experts who traced the contours of all objects within this volume (see Figure 1C). These segmentations were converted into desired affinity graphs, as described in section 3, and half the volume was used for training and the other half for testing. While we refer to human tracings as ground truth, it should be noted that there is disagreement among experts on the exact placements of boundaries. This variability is used in our analysis to suggest a baseline interhuman error rate. Figure 1C highlights some of the difficult regions typical of those encountered in these data (red arrows). In z section 22 (upper

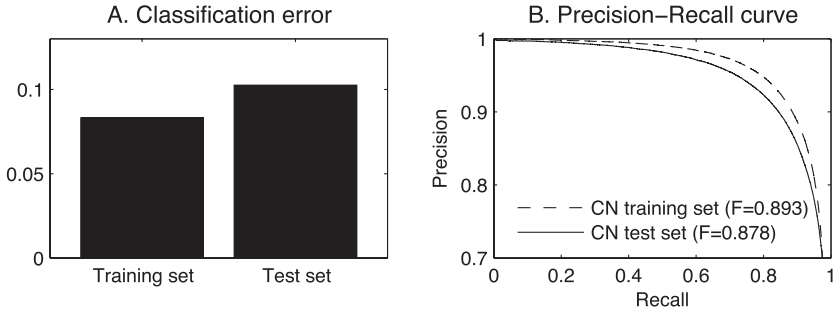

Figure 4: Performance of the convolutional network at predicting affinity between individual voxels. (A) Approximately 10% of the affinities are misclassified in the test set. (B) A more thorough quantification of the continuous-valued CN output using precision-recall curves (see appendix C) as in Martin et al. (2004) shows good performance (higher F-scores and curves closest to the upper right are superior).

panel), the boundary between two objects is very faint, which could lead to an incorrect merge of these processes. Section 38 (lower panel) shows objects packed closely together with small cross-sections, which can be as few as four voxels per section. These processes could be split by incorrect labeling of only a few voxels. Since such splits and mergers are important properties of the resulting segmentations, we have developed metrics to quantify them (see appendix C).

4.2 Convolutional Networks Can Be Trained to Produce High-Quality Affinity Graphs. Convolutional networks are trained using backpropagation learning to take an EM image as input and output three images representing the desired affinity graph. The gradient descent procedure is detailed in section B.1.

Our CN has an architecture with three hidden layers, each containing six feature maps. All filters in the CN are of size $5 \times 5 \times 5$, but since CN contains four convolutions between input and output, a single output voxel is a function of a $17 \times 17 \times 17$ voxel region of the input image.

After gradient-based training, the performance of the network at generating affinities can be quantified by measuring the classifier error at each edge as compared with target affinities generated from the human segmentations. The optimal threshold for classification is chosen by optimization with respect to the training set. To demonstrate generalization, we quantify these results on both the training and test image volumes. The classifier error on this test image is quantified in Figure 4A, showing that approximately 90% of the edges are classified correctly. Figure 4B quantifies the performance at edge classification using a precision-recall curve

(Van Rijsbergen, 1979; Martin, Fowlkes, & Malik, 2004). The curve shows how the classification performance changes as the threshold is varied from a high value (left) to a low value (right). The definitions of precision and recall are given in section C.1. Here it is enough to know that perfect performance would be a curve close to the upper and right boundaries of the graph. The precision-recall curve is sometimes summarized by a single number, the F-score, which equals 1 for perfect performance. As in Martin et al. (2004), we present the precision-recall curve for classification of the rarer class of boundary edges. In our representation, this corresponds to the precision and recall of negative examples.

From Figure 4B, we can see that the absolute performance of the CN network is encouraging. This demonstrates that it is possible to apply machine learning to image segmentation without using hand-selection of features. The CN has an input dimensionality of 4913 voxels and 12,021 free parameters. In spite of the large number of parameters, there appears to be little overfitting, given that performance on the test set is comparable to the training results. The lack of overfitting may be due to the large size of the training set—about half a megavoxel. This could be viewed as half a million examples of the affinity graph generation task, though this estimate is a bit misleading since the examples are not statistically independent. In contrast, other machine learning approaches to segmentation have relied on specially selected features and used learning only to optimize the combinations of these features. For example, Fowlkes et al. (2003) use classifiers with seven inputs and only a few dozen free parameters. Our approach shows that good performance can be achieved without the careful design and selection of image features.

4.3 Affinity Graphs Generated Using CN Improve Segmentation Performance. In the previous section, we quantified network performance by measuring the ability to generate the desired affinity graph. However, the affinity graph is not an end in itself, only an intermediate step toward a segmentation. Therefore, we evaluated the CN affinity graphs by applying the popular normalized cuts algorithm to partition the affinity graph into segments (see section B.2). It bears repeating, however, that our affinity graph can be paired with any graph-partitioning algorithm.

Figure 5 shows that the segmentations using normalized cuts (lower left) qualitatively match the human tracings (upper right). To more thoroughly quantify the segmentation performance, we measured the number of split and merge errors made in our segmentation. These results are shown in Figure 6. In comparing the segmentation errors of the normalized cuts algorithm using the standard affinity with the CN affinity (NCUT-STD versus NCUT-CN), we see a striking reduction of an order of magnitude in the number of splits and a several-fold improvement in the number of splits. Because two independent human labelings are available, we can compare human versus human segmentations with the same metrics (Human-Human).

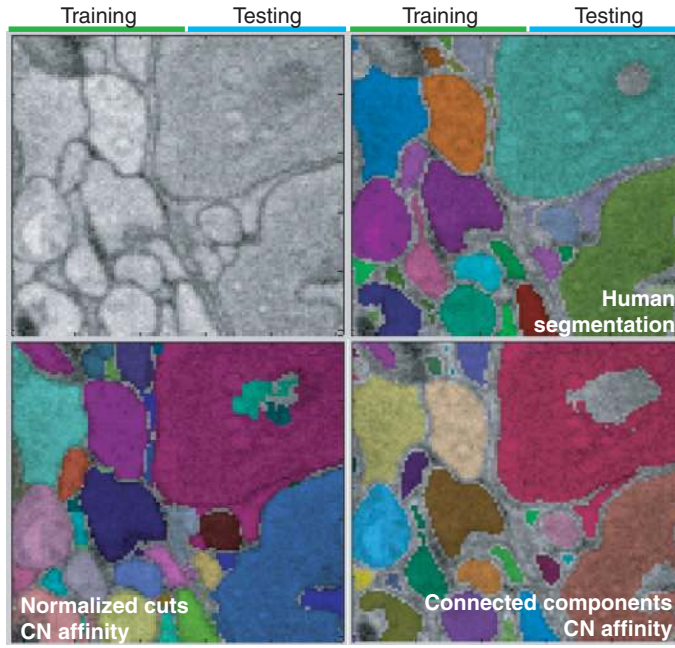

Figure 5: (Top row) Original EM images and human-labeled segmentation, where different colors correspond to different segments. (Bottom left) Using normalized cuts on the output of the CN network qualitatively segments many of the objects correctly. (Bottom right) The connected components procedure using CN affinity creates segmentations that are structurally superior to normalized cuts with fewer splits and mergers, and the segments are shrunk within the intracellular regions.

Note that normalized cuts require the selection of the number of segments in an image, k. The true number of segments is difficult to determine ahead of time and scales with the image size in an unpredictable manner. This is known to be a difficult parameter to optimize, and for our experiments, we choose k to be the known true number of segments to demonstrate the best possible performance of this algorithm, although this information would typically be unavailable at test time.

4.4 CN Affinity Graph Allows Efficient Graph Partitioning Using Connected Components. A significant disadvantage of the normalized cuts algorithm is its considerable computational cost. As described in section B.2, the run time is typically at least quadratic in the image size, and our images contain a large number of voxels. There is a simpler alternative to normalized cuts that does not require the solution of a complex

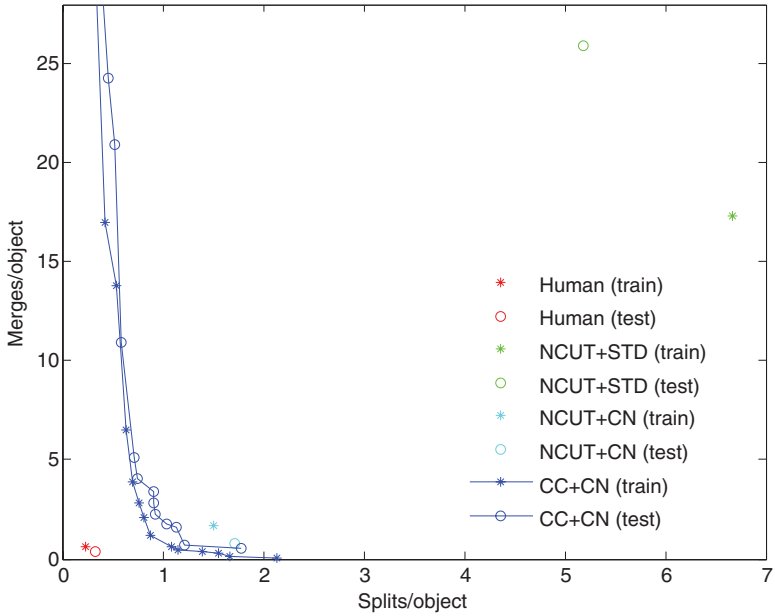

Figure 6: Segmentation performance is quantified by measuring the number of splits and merges per true segment. Points closer to the origin have lower segmentation error. Circles and asterisks are used to denote the test and training set errors, respectively. CC+STD has split-merge errors of (121.9,15.7) and (97.9,13.2) on the training and test set, respectively and is omitted here for clarity. Many different segmentations can be generated by varying the threshold parameter for the CC algorithm, yielding under- to oversegmentation.

optimization problem. The affinity graph is first pruned by removing all edges with affinities below a certain threshold. Then a segmentation is generated by finding the connected components (CC) of the pruned graph, where a connected component is defined as a set of vertices that can be reached by following a path of connected edges (Cormen, Leiserson, Rivest, & Stein, 2000). Since the connected components can be found in run time roughly linear in the number of voxels, this approach is significantly faster. Unlike with the normalized cuts procedure, the number of objects need not be known ahead of time, since the adjustable threshold parameter is independent of the number of objects.

This algorithm is extremely sensitive to errors in the affinity graph since a single misclassified link can alter a segmentation by merging objects. In practice, we find that our learned affinity graph is accurate enough that this rarely happens. In contrast, the hand-designed affinity graph is extremely noisy and leads to poor segmentations using this partitioning algorithm.

Varying the threshold parameter for binarizing the graph results in segmentations ranging from over- to undersegmentation of the image. The performance of this segmentation algorithm using the CN affinity can be seen for various thresholds in Figure 6. The performance of the standard affinity using connected components is too poor to fit in this plot. It is notable that the simple connected components strategy outperforms the more sophisticated normalized cuts graph-partitioning algorithm.

Visualizations of typical components are shown in Figure 7, comparing the original human segmentation to CN+CC and CN+NCUT. In Figure 7A, the large branching component is segmented correctly by both algorithms, while in Figure 7B, both algorithms split the component at narrow regions (CC preserves slightly more of the object), and in Figure 7C, the thin branching component is correctly segmented.

5 Discussion

In this section, we discuss existing machine learning approaches to image segmentation and how our approach differs. In general, most existing approaches use extensive pre- and postprocessing with learning as a middle step. We show that emphasizing the learning stage can lead to an efficient segmentation algorithm with simple and computationally efficient pre- and postprocessing.

As an example, the problem of segmenting retinal blood vessels is an important medical application for which many machine learning approaches have been implemented (Ricci & Perfetti, 2007; Sinthanayothin et al., 1999). Most methods can be characterized as using edge detectors or wavelet features followed by a classifier that produces a binary vessel or no-vessel decision at each pixel and postprocessing to remove small spurious segments. Sinthanayothin et al. (1999) use a neural network on image patches augmented with Canny edge-detected versions of the same patches. They argue qualitatively that relying less on the learned network and more on pre- and postprocessing makes the computation more efficient and relies on fewer training data. We essentially take an opposite view: relying more on the network avoids hand-designed features, which may throw away data or bias results. In other work, Ricci and Perfetti (2007) use a hand-designed set of thin, oriented line detectors, as well as local average intensities, as features for input to an SVM classifier. The classifier outputs are vessel or no-vessel decisions at each pixel location in the image. Preprocessing for these methods generally includes a local adaptive contrast equalization method specifically designed for these retinal images. In contrast, our method uses only minimal preprocessing and relies on the learning procedure to account for contrast and other image variations. While our method may require a larger training set to include a statistically representative set of image variations, the learned convolutional filters are directly adapted to improve the

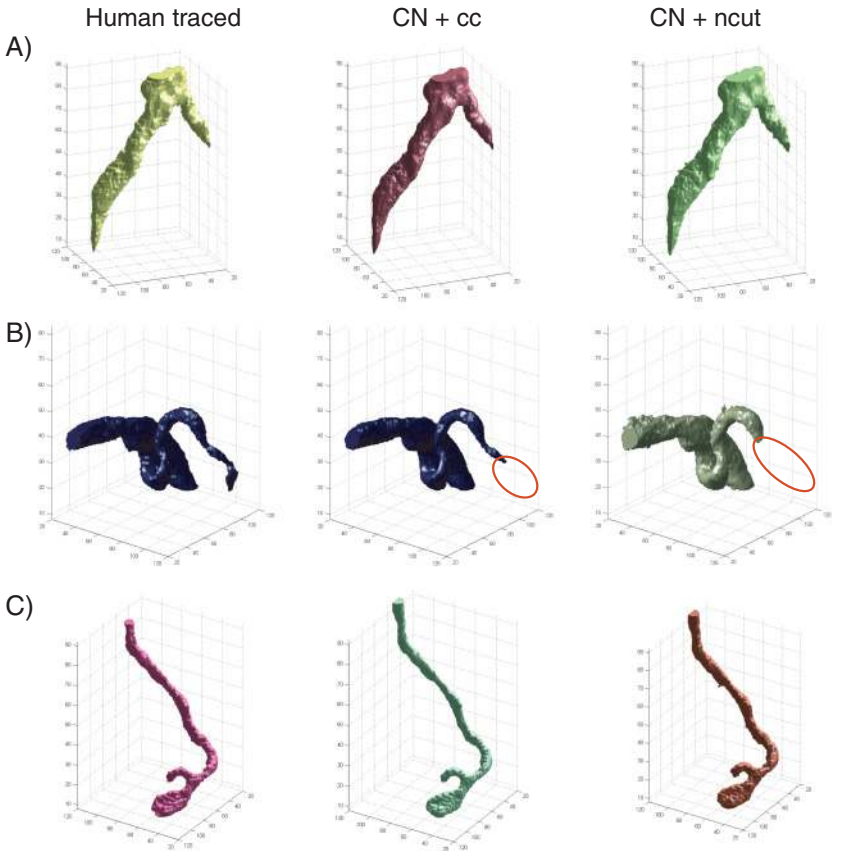

Figure 7: Components traced by human (left) compared with automated results, convolutional network with connected component segmentation CN-CC (middle) and normalized cut segmentation CN-NCUT (right). (A) Correctly reconstructed object, showing close agreement between human tracings and algorithm output. (B) An object that has thin processes incorrectly split by the algorithms. A larger part of the object is split by normalized cuts than connected components. The red oval indicates the size of the process that was split. (C) Both algorithms correctly segment a thin branching process.

target metric and so are less likely than hand-designed preprocessing to throw away valuable information.

To appreciate the power of the convolutional network (CN), it is useful to compare it to a standard multilayer perceptron neural network (NN), which has long been used to classify image patches (Egmont-Petersen et al., 2002; Sinthanayothin et al., 1999). Conceptually the main difference between a CN and an NN is the fact that an NN has only one stage of spatial filtering,

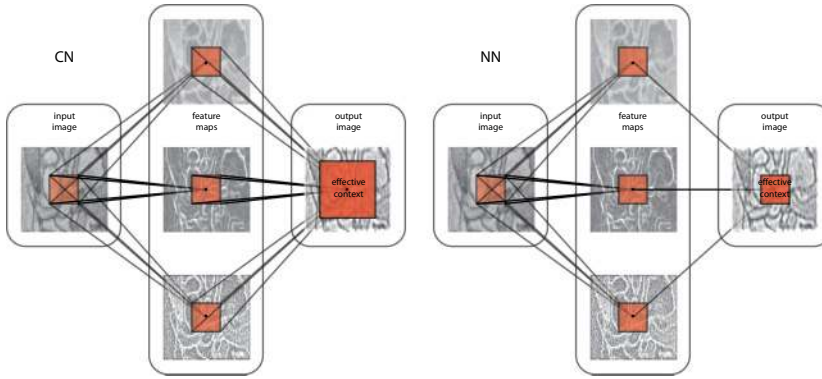

Figure 8: A convolutional network constructs progressively larger as well as more complex image features. A neural network can construct only more complex features at each layer, not larger features.

while a CN can incorporate filtering at every layer of the network. This is a subtle difference, however, since patch-based NNs can be constructed that are constrained in a way that makes them exactly equivalent to a given CN, and vice versa. We explore the details of this relationship in appendix C. The practical benefit of using a CN is that it is easier to codify and implement the spatial constraints that are relevant to an image processing problem.

We can compare an unconstrained patch-based NN to a CN to understand the potential benefits of a CN. The unconstrained patch-based NN is a special case of the CN, where filters in all layers after the first are scalars. In this NN, the image context used to make classification is fixed to be the size of the filters in the first layer, regardless of the number of layers in the network. In contrast, as sketched in Figure 8, the image context of a CN grows with every additional layer in the network. This suggests the capability of each layer in the CN to construct larger spatial features using smaller spatial features detected by the previous layer.

5.1 Efficient Graph Partitioning. There has been much research in recent years on developing good graph-partitioning criteria and algorithms (Shi & Malik, 2000; Felzenszwalb & Huttenlocher, 2004; Boykov, Veksler, & Zabih, 2001; Cour, Benezit, & Shi, 2005; Gdalyahu, Weinshall, & Werman, 1999). Continuing our theme of relying more on learning methods and less on elaborate postprocessing, we have instead focused on a method for generating better affinity graphs. We have shown that convolutional networks can learn an affinity graph (directly from image patches) that is good enough to be segmented without the need for complex graph partitioning such as normalized cuts. While it is reasonable to expect that normalized cuts would not be optimal for segmenting neurons, it is surprising that the

simpler connected components algorithms work this well. This points to the degree to which the learned affinity graph simplifies the segmentation problem.

A consideration of the normalized cut objective suggests why it has trouble with our data set. The normalized cuts criterion specifically aims to find segments with minimal surface-area-to-volume ratios, while neurons are highly branched structures with extremely high surface area. This makes the criterion ill suited to our problem. Also, normalized cut is biased toward segments of roughly equal size. Because the objects in our database vary in volume over two orders of magnitude, this bias is not helpful. This illustrates the danger of using incorrect assumptions in an image segmentation application and the advantages of adapting the algorithm to the application by using training data.

In contrast to existing methods where affinities are computed based on a few hand-picked features, our approach is considerably more powerful in that it learns both the features and their transformation. This allows us to learn affinity graphs for applications where there is little preexisting knowledge of good features. This added flexibility comes at the price of training a classifier with many parameters. But our results show that such a classifier can be trained effectively using a CN.

Interestingly, our good segmentation performance is found using a graph with only nearest-neighbor edges. One could argue that better performance might be achieved by learning a graph with more edges between voxels. Indeed, researchers have found empirically that segmentation performance using hand-designed affinity functions can be improved by adding more edges (Cour et al., 2005). The extension of our learned affinity functions to more edges is left for future work.

5.2 Progress Toward Neural Circuit Reconstruction Using Serial-Section EM. We have made a first step toward automated reconstructing of neural circuits using images from serial block-face scanning electron microscopy (SBF-SEM; Denk & Horstmann, 2004). Figure 9 shows the 3D reconstruction of selected neural processes in the image volume, confirming that the generated segments are biologically realistic. Figure 10 elaborates on this by showing the largest 100 components in the $320 \times 200 \times 100$ volume (omitting large glia and cell bodies). Most of the largest 40 components span the volume, and smaller components appear to either terminate within the volume or be fragments of glial-like processes. While we have observed good performance with our current segmentation algorithm, significant progress must yet be made in order to achieve large-scale neural reconstruction. In particular, while performance is close to human level on voxel error and merged objects, there are significantly more split objects in our network output. This indicates that there are small, thin processes that are broken and could be reconnected using heuristics or another higher-level learning procedure. We have yet to make use of high-level segmentation cues such

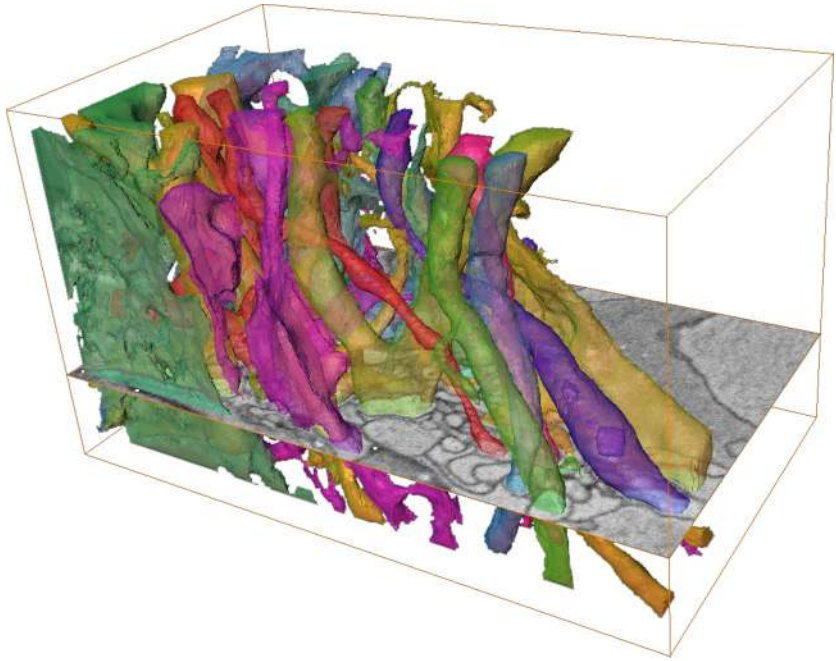

Figure 9: Three-dimensional reconstructions of selected processes using our algorithm (CN+CC). Large cell bodies are not shown to avoid hiding smaller processes.

as the identity and shapes of the neurons in our data set. Efficiently representing and integrating such high-level information with low-level affinity decisions is the next challenge.

Appendix A: Relationship Between Convolutional and Neural Networks

A.1 Neural Networks and Image Processing. Conventionally NNs are classifiers that operate on vector-valued input to produce scalar or vector-valued output predictions. Similar to a CN, the NN is described using a graph. But while the nodes and edges of a CN are image valued, those of a NN are scalar valued. And the values of the nodes x_a in an NN are computed as $x_a = f(\sum_b W^{ab} x_b + h_b)$, where W^{ab} and h_b are now scalars. Despite the differences, there is an equivalence between these two architectures in the context of image processing.

A neural network may be applied to an image processing problem in several ways. In the simplest case, the pixels of an entire image may be reordered as a vector and provided as input to a NN. This approach lends itself well to tasks such as image recognition, where the input is image

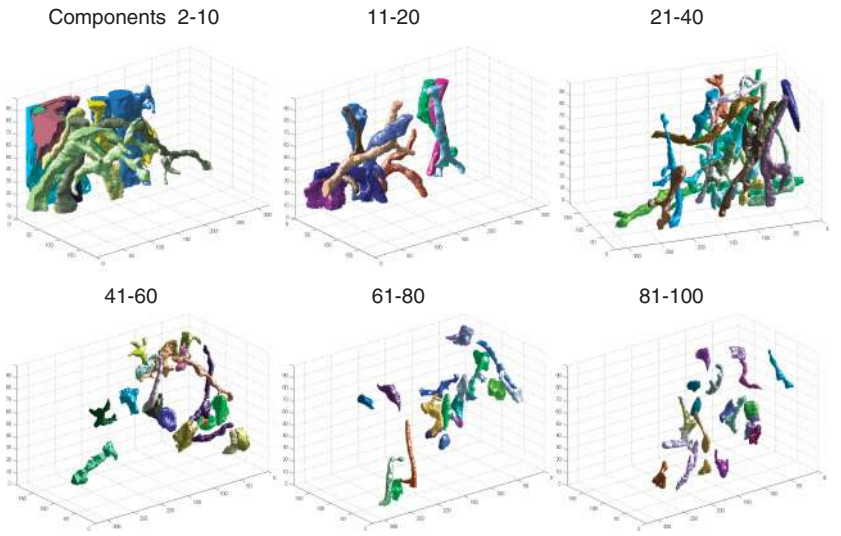

Figure 10: The 100 largest components in a $320 \times 200 \times 100$ volume ($7.86\ \mu\text{m} \times 5.24\ \mu\text{m} \times 5\ \mu\text{m}$), omitting cell bodies and large glial processes for clarity.

valued but the output is scalar valued or vector valued. A second approach is more suitable to nonlinear filtering applications where we wish both the input and the output to be image valued. The translational invariance required by a filtering application is achieved by applying an NN to overlapping image windows. The image-valued output is generated by pasting together scalar-valued NN output from translated windows of the image. We call this a *patch-based* NN. The network labeled "NN" in our experiments is exactly such a network, and a cartoon of it is shown in Figure 8.

A.2 Patch-Based Neural Networks Are a Special Case of Convolutional Networks. Using the terminology developed in section 2, we can represent any patch-based NN classifier as a special case of CNs. Let us recall the standard forms of the neural network and convolutional network equations,

$$x_\alpha = f\left(\sum_\beta W^{\alpha\beta} x_\beta + h_\alpha\right)$$

$$I_a = f\left(\sum_b w_{ab} I_b + h_a\right),$$

where x_α, x_β, $W^{\alpha\beta}$, and h_α are scalars; I_a, I_b are images; w_{ab} is a filter; and h_a is a scalar. In the case of the first layer of a patch-based NN, the elements of

x_β form the pixels in the image patch. But since the output of a patch-based NN is constructed by translating the network, there is an exact equivalence between convolutional filtering and linear matrix multiplication implied in this equation for the first layer of the NN. This means each row $W^{\alpha\cdot}$ of the weight matrix can be exactly interpreted as the filter w_{ab} applied to the input image I_b. This implies a pairing between I_a and x_α. Each image in the second layer I_a is constructed by pasting together the values of x_α derived from translating the image patch.

After the first layer, all further computation in an NN occurs through simple linear combinations of the existing features x_α. Thus, the convolutional network implied by a patch-based NN performs no filtering after the first layer. The "filters" w_{ab} in the later layers correspond exactly to scalars given by $W^{\alpha\beta}$.

Thus, the crucial difference between an NN and a CN is in the nature of the w_{ab} parameters. An NN restricts all w_{ab} filters not operating directly on the input image to be scalars. This means that only the first layer of computation involves image convolutions, while the rest are constrained to be simple linear combinations of existing image features I_a, followed by a nonlinearity. A CN may be seen as a more powerful generalization of an NN in a manner appropriate for image processing applications since convolutions are allowed at all stages in the network.

A.3 Convolutional Networks Are a Special Case of Neural Networks. A CN can also be seen as a special case of a NN with vectorial output by recalling that convolution is a linear operation that can be represented in terms of matrix multiplication by a special "convolution matrix." In this view, I_a is an image represented as a vector, and W^{ab} is the convolution matrix representing convolution by the filter w_{ab}:

$$I_a = f\left(\sum_b W^{ab} I_b + h_a\right). \tag{A.1}$$

Since a convolution matrix M may be constructed from a filter w as $M_{ij} = w_{i-j}$, we can see that the weight matrix of an NN equivalent to a CN has a constrained translation-invariant structure that is useful for image processing. However, the notation, interpretation, and representation of the weight matrix and corresponding hidden units become complicated and less useful for a CN with more than one hidden unit in each layer.

A.4 Comparing CN and NN for Affinity Graph Generation. In order to assess the advantages of CN over NN, we also trained a CN with NN constraints on the task of affinity graph generation. This network (see Figure 11) has a single hidden layer of 75 feature maps. The filters connecting the input image to the feature maps in the hidden layer have size

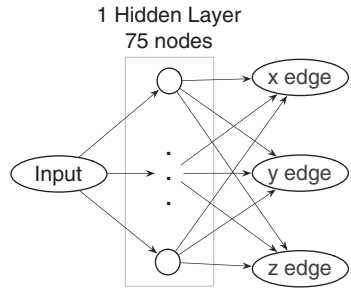

Figure 11: A restricted convolutional network that is equivalent to a two-layer neural network. Only the first layer of the network is convolutional; edges from the input image to the hidden nodes represent convolution filters, while edges from the hidden units to the output units are scalar valued. This network has many more nodes in the hidden layer than our CN but fewer layers.

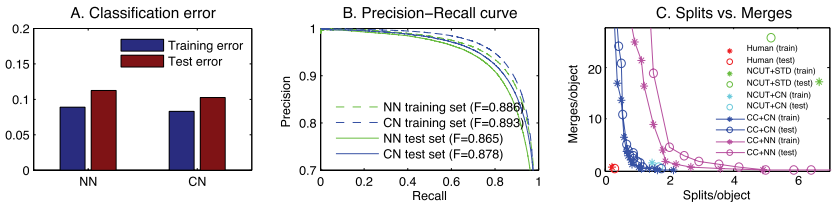

Figure 12: CN and NN have similar performance at affinity prediction, but CN has superior segmentation performance. (A) Approximately 10% of the affinities are misclassified in the test set. (B) A more thorough quantification of the continuous-valued affinity output using precision-recall curves (see section C.1) as in Fowlkes et al. (2003) shows good performance (higher F-scores and curves closest to the upper right are superior). (C) CN outperforms NN with fewer split and merge errors (curves closest to the origin are superior).

$7 \times 7 \times 7$. The filters connecting the feature maps in the hidden layer to the output images are $1 \times 1 \times 1$ scalars. Since these filters are scalars, each output image is a linear combination of the images in the hidden layer after passing through a sigmoid nonlinearity. In contrast to our CN, this network requires more than twice as many free parameters (25,876) to process an input patch of only $7 \times 7 \times 7$. Our attempts to train NN-like networks with larger filters were plagued by poor generalization performance and with gradient learning optimizing to poor local minima in parameter space.

From Figure 12, we can see that the CN is slightly superior to the NN on the training set. Since the NN has twice as many parameters as CN, one might expect it to be a more powerful representation and achieve lower training error. However, the CN uses a larger context than the NN ($17 \times 17 \times 17$ versus $7 \times 7 \times 7$), and this may help the CN achieve lower training error.

In addition to the improvement in performance, after training has been completed, the CN is about twice as fast as the NN at the task of generating an affinity graph. The run time is roughly proportional to the number of parameters, and the NN has twice as many as the CN. More important CN makes fewer split-and-merge errors, leading to superior segmentation performance.

While both the CN and NN can learn the affinity graph generation task without hand selection of features, the CN achieves both superior speed and accuracy compared to the NN. This is probably because the CN representation is more efficient for image processing tasks than the NN. The difference between the CN and the NN on this data set is not large and therefore may not be statistically significant. However, the differences in affinity prediction as well as segmentation performance seem to be larger in preliminary investigations using larger data sets not reported here.

Appendix B: Segmentation Algorithms

B.1 Gradient Learning by Backpropagation. We use the backpropagation procedure detailed by LeCun et al. (1989) to train our CN. To speed up convergence of the gradient procedure, we used some simple tricks from LeCun, Bottou, Orr, & Muller (1998), which we detail here.

The CN used in this letter corresponds to the directed graph in Figure 2, where the nodes represent images and the edges represent filters. It contains three hidden layers, each with six feature maps. Each feature map is constructed according to equation 2.1 by filtering the images corresponding to incoming nodes with three-dimensional filter kernels of size $5 \times 5 \times 5$. Thus, running the network to produce the nearest-neighbor affinity graph requires 96 separate convolutions.

We initialize all the elements of the filters w_{ab} and the biases h_a randomly from a normal distribution of standard deviation $\sigma = 1/\sqrt{|w| \cdot |b|}$, where $|w|$ is the number of elements in each filter (here, 5^3) and $|b|$ is the number of input feature maps incident on a particular output feature map. This choice mirrors a suggestion in LeCun, Bottou, Orr, & Muller (1998) that weight vectors have unit norm.

We trained the randomly initialized CN using the iterative optimization procedure of stochastic gradient descent with diagonal rescaling. On each iteration, a mini-batch was generated by randomly sampling $6 \times 6 \times 6$ image patches from the training image. Backpropagation was performed to compute the gradient, which was then rescaled by a diagonal approximation to the Hessian as in LeCun, Bottou, Orr, & Muller (1998) with a learning rate of 10^{-4}. Training was stopped after 80 epochs were performed, as the training error had plateaued. There was rarely need for cross-validation as little overfitting was observed.

B.2 Normalized Cuts. The multiway normalized cut problem is posed as finding the partition of a graph $\mathcal{G}$ into k groups of nodes $\mathcal{V}_i$ such that the following objective function is minimized:

$$NCut(\{\mathcal{V}_1, \ldots, \mathcal{V}_k\}) = \min_{\mathcal{V}_1,\ldots,\mathcal{V}_k} \sum_{i=1}^{k} \frac{\text{links}(\mathcal{V}_i, \mathcal{G}\backslash\mathcal{V}_i)}{\text{degree}(\mathcal{V}_i)}, \tag{B.1}$$

where $\mathcal{G}\backslash\mathcal{V}_i$ is the set of vertices not in $\mathcal{V}_i$. The number links$(\mathcal{V}_i, \mathcal{G}\backslash\mathcal{V}_i)$ are usually referred to as the *cut value* since they represents the sum of all the edges cut due to the partition $\mathcal{V}_i$. Here, the cut value is normalized by the association of this region given by degree$(\mathcal{V}_i)$ (which is the sum of all edges between nodes in $\mathcal{V}_i$). Roughly speaking, links$(\mathcal{V}_i, \mathcal{G}\backslash\mathcal{V}_i)$ is proportional to the surface area of of the region $\mathcal{V}_i$, while degree$(\mathcal{V}_i)$ is proportional to the volume. So normalized cuts may be thought of as generating partitions with minimal surface-area-to-volume ratios.

Since the problem of optimizing equation B.1 is known to be NP-hard, Shi and Malik (2000) suggested a polynomial time approximation based on spectral methods. This algorithm was prohibitively slow and memory intensive for our application, because a k-way segmentation requires finding at least $k + 1$ eigenvectors of a very large matrix. Instead, we used the fast multiscale kernel weighted k-means algorithm, which finds local minima of the normalized cuts cost function (Dhillon, Guan, & Kulis, 2005). For large problems such as ours (more than 100,000 nodes), this algorithm is significantly faster than the spectral method and achieves better results.

We found that superior segmentations were obtained if the affinity graph was first coarsened by finding atomic regions. This is a standard procedure in segmentation algorithms where strongly connected regions are grouped together to form super-pixels. Edges between the super-pixels now correspond to the sum of all affinities between original nodes in each super-pixel.

When algorithm was run, the desired number of segments, k, was set to the true number of segments, which was known in the training and test sets. While this information would not be available in practice, it was used here to optimize the performance of normalized cuts.

Appendix C: Performance Metrics

We detail the performance metrics used to measure performance: the precision-recall method used to evaluate affinity-graph learning and the split-merger counts for evaluating segmentation performance.

C.1 Precision-Recall Curves. The precision-recall curve is constructed by computing the values of precision and recall for all values of the classifier threshold, and this methodology has been used to evaluate segmentation algorithms (Martin et al., 2004). Precision measures the fraction of examples

classified positive that are truly positive, while recall measures the total fraction of positive examples that are classified positive. A good classifier would have a precision close to 1 for all values of recall except when recall equals 1. The curve can be characterized by a single F-score, which is the maximum of the weighted harmonic mean of precision p and recall r over the curve, $F = \max \; pr/(\alpha p + (1-\alpha)r)$, where here the weighting factor is $\alpha = 0.5$. Higher F-scores indicate better performance. An advantage of the precision-recall measure is that it can emphasize performance on the rarer of the two classes.

Note that for consistency with Martin et al. (2004), we compute precision-recall for the detection of boundaries. In our affinity representation, boundaries correspond to negative examples. Hence, in our case, we compute precision and recall for the detection of negative examples by inverting the classification.

There are two important differences between our evaluation and that of Martin et al. (2004). First, we measure precision-recall curves for affinity graphs rather than boundary detection. Second, we do not perform any matching procedure, such as the bipartite boundary matching in Martin et al. (2004) to improve the correspondence between our affinity graph and the ground-truth affinity graph. In these differences, our measures are more similar to those used by Fowlkes et al. (2003).

C.2 Splits and Mergers. The number of splits and mergers caused by the algorithm is a useful measure of segmentation performance. Split errors are quantified as the number of fragments an object in the ground truth data set is broken into. Merge errors are quantified simply as the number of times objects in the ground truth were erroneously connected to each other. These quantities are computed as a function of two segmentations: the ground truth and a test segmentation.

A bipartite graph is constructed where the nodes of the graph fall into two groups. The first group of nodes represents the objects in the ground-truth segmentation, with one node for each object. Similarly, the second group of nodes represents the objects in the test segmentation. Edges in this bipartite graph are undirected and unweighted and drawn between a ground-truth node and a test node if they claim a common region in the image. This graph now represents the overlap between objects in each segmentation. This overlap graph can be used to compute the number of splits and merges in the test segmentation. A perfect segmentation with no splits or merges would result in a one-to-one correspondence of nodes in the overlap graph, with each node having exactly one edge. Splits and mergers result in changes of the degree of the overlap graph.

A split is said to occur when an object in the ground truth is erroneously broken into more than one object in the test segmentation. In the overlap graph, this corresponds to a situation where one node in the ground truth has edges to more than one node in the test segmentation. The number of

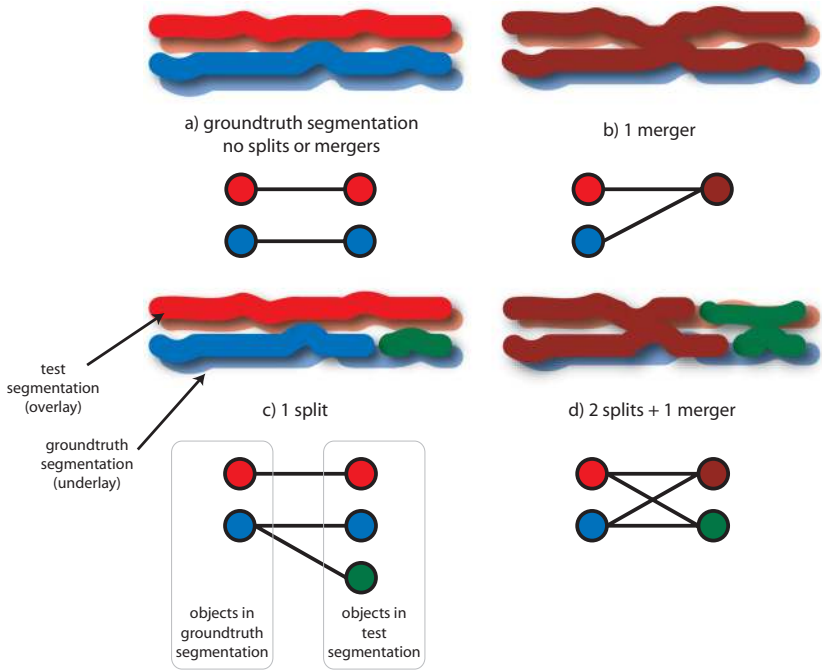

Figure 13: Bipartite graph representing splits and mergers. Test segmentation is overlaid on the ground-truth segmentation, and the overlap between objects is measured. Overlap codified as a bipartite graph is shown for the example cases of (a) perfect segmentation, (b) merger, (c) split, and (d) two splits and a merger.

splits corresponds to the number of edges of a ground-truth node in excess of 1. This may be computed by subtracting the number of ground-truth nodes with edges, from the total number of edges in the bipartite graph.

A merge is said to occur when an object in the ground truth is erroneously joined with another ground-truth object. This happens when a test object overlaps with more than one ground-truth object. In the overlap graph, this corresponds to nodes in the ground truth becoming connected via nodes in the test segmentation. A merge has occurred between a pair of ground-truth objects if there is at least one node in the test segmentation that has edges to both ground-truth nodes. To compute the number of merges, we first compute the distance matrix for all pairs of ground-truth nodes. The number of merges corresponds to the number of pairs of ground-truth nodes with distance equal to 2. This implies that they are connected by the shortest possible path in the bipartite graph via one test node. For a small fraction of pairs of objects, there may be more than one such shortest path corresponding to situations as depicted in Figure 13. Here a coincidence

of splits and merges leads to the same two objects being merged in two different locations. We choose to count such merges only once since the same two objects are involved.

It should be noted that the measurement of splits and mergers is agnostic to variations in the number of voxels in the estimated and ground-truth objects. Therefore, a neurite with an extra voxel or a missing voxel would not contribute to split or merge errors. A missing voxel that does not lead to the splitting of a neurite (in the case of an extremely narrow neurite) causes no errors, and an extra voxel is not measured as a merge error as long as it does not lead to the merging of two neurites. Thus, the split-merge metric is robust to minor variations in the sizes of segments.

References

Al-Kofahi, K., Lasek, S., Szarowski, D., Pace, C., Nagy, G., Turner, J., et al. (2002). Rapid automated three-dimensional tracing of neurons from confocal image stacks. *IEEE Trans. Info. Tech. Biomed.*, *6*(2), 171–187.

Andres, B., Köthe, U., Helmstaedter, M., Denk, W., & Hamprecht, F. A. (2008). Segmentation of SBFSEM volume data of neural tissue by hierarchical classification. In G. Rigoll (Ed.), *Pattern recognition* (vol. 5096 of *LNCS*, pp. 142–152). Berlin: Springer.

Boykov, Y., Veksler, O., & Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *23*(11), 1222–1239.

Briggman, K. L., & Denk, W. (2006). Towards neural circuit reconstruction with volume electron microscopy techniques. *Curr. Opin. Neurobiol.*, *16*(5), 562–570.

Carlbom, I., Terzopoulos, D., & Harris, K. M. (1994). Computer-assisted registration, segmentation, and 3D reconstruction from images of neuronal tissue sections. *IEEE Trans. Medical Imaging*, *13*(2), 351–362.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2000). *Introduction to algorithms*. Cambridge, MA: MIT Press.

Cour, T., Benezit, F., & Shi, J. (2005). Spectral segmentation with multiscale graph decomposition. *IEEE Conf. Computer Vision and Pattern Rec.* (Vol. 2, 1124–1131). Piscataway, NJ: IEEE.

Denk, W., & Horstmann, H. (2004). Serial block-face scanning electron microscopy to reconstruct three-dimensional tissue nanostructure. *PLoS Biol.*, *2*(11), e329.

Dhillon, I., Guan, Y., & Kulis, B. (2005). A fast kernel-based multilevel algorithm for graph clustering. In *Proc. ACM SIGKDD Intl. Conf. Knowledge Discovery in Data Mining* (pp. 629–634). New York: ACM Press.

Egmont-Petersen, M., de Ridder, D., & Handels, H. (2002). Image processing with neural networks: A review. *Pattern Recognition*, *35*(10), 2279–2301.

Felzenszwalb, P., & Huttenlocher, D. (2004). Efficient graph-based image segmentation. *International Journal of Computer Vision*, *59*(2), 167–181.

Fiala, J. C. (2005). Reconstruct: A free editor for serial section microscopy. *Journal of Microscopy*, *218*(1), 52–61.

Fowlkes, C., Martin, D., & Malik, J. (2003). Learning affinity functions for image segmentation: Combining patch-based and gradient-based approaches. . *Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Washington, DC: IEEE Computer Society.

Gdalyahu, Y., Weinshall, D., & Werman, M. (1999). Stochastic image segmentation by typical cuts. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Vol. 2, pp. 596–601). Washington, DC: IEEE Computer Society.

Helmstaedter, M. N., Briggman, K. L., & Denk, W. (2007). Segmentation of SBFSEM volume data of neural tissue by hierarchical classification. *Society for Neuroscience Abstracts*, pp. 142–152.

Jain, V., Murray, J., Roth, F., Turaga, S., Zhigulin, V., Briggman, K., et al. (2007). Supervised learning of image restoration with convolutional networks. In *Proc. Intl. Conf. Computer Vision*. Piscataway, NJ: IEEE.

Jurrus, E., Hardy, M., Tasdizen, T., Fletcher, P., Koshevoy, P., Chien, C.-B., et al. (2009). Axon tracking in serial block-face scanning electron microscopy. *Medical Image Analysis, 13*, 180–188.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., et al. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, *1*, 541–551.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

LeCun, Y., Bottou, L., Orr, G., & Muller, K. (1998). Efficient backprop. In G. Orr & K. Muller (Eds.), *Neural networks: Tricks of the trade*. Berlin: Springer.

Liang, J., McInerney, T., & Terzopoulos, D. (2006). United Snakes. *Medical Image Analysis*, *10*(2), 215–233.

Martin, D. R., Fowlkes, C. C., & Malik, J. (2004). Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, *26*(1), 1–20.

Mishchenko, Y. (2009). Automation of 3D reconstruction of neural tissue from large volume of conventional serial section transmission of electron micrographs. *Journal of Neuroscience Methods*, *176*, 276–279.

Nekovei, R., & Sun, Y. (1995). Back-propagation network and its configuration for blood vessel detection in angiograms. *IEEE Trans. Neural Networks*, *6*(1), 64–72.

Ning, F., Delhomme, D., LeCun, Y., Piano, F., Bottou, L., & Barbano, P. E. (2005). Toward automatic phenotyping of developing embryos from videos. *IEEE Transactions on Image Processing*, *14*(9), 1360–1371.

Ricci, E., & Perfetti, R. (2007). Retinal blood vessel segmentation using line operators and support vector classification. *IEEE Transactions on Medical Imaging*, *26*(10), 1357–1365.

Shi, J., & Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*(8), 888–905.

Sinthanayothin, C., Boyce, J. F., Cook, H. L., & Williamson, T. H. (1999). Automated localisation of the optic disc, fovea, and retinal blood vessels from digital colour fundus images. *British Journal of Opthalmology*, *83*, 902–910.

Smith, S. (2007). Circuit reconstruction tools today. *Current Opinion in Neurobiology*, *17*(5), 601–608.

Van Rijsbergen, C. (1979). *Information retrieval*. Burlington, MA: Butterworth-Heinemann.

Vasilevskiy, A., & Siddiqi, K. (2002). Flux maximizing geometric flows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*(12), 1565–1578.

White, J., Southgate, E., Thomson, J., & Brenner, S. (1986). The structure of the nervous system of the nematode *Caenorhabditis elegans*. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, *314*(1165), 1–340.

Received October 9, 2008; accepted May 14, 2009.