Learning image segmentation and hierarchies by learning ultrametric distances

by

Srinivas C. Turaga

B.S., Chemistry, University of Massachusetts, Amherst (2002)
B.S., Computer Science, University of Massachusetts, Amherst (2002)

Submitted to the Department of Brain & Cognitive Sciences
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computation

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2009

© Massachusetts Institute of Technology 2009. All rights reserved.

Author . . . .
Department of Brain & Cognitive Sciences
August 14, 2009

Certified by. . . . .
H. Sebastian Seung
Professor of Computational Neuroscience; Investigator, Howard Hughes
Medical Institute
Thesis Supervisor

Accepted by . . . .
Earl K. Milller
Chairman, Department Committee on Graduate Theses

Learning image segmentation and hierarchies by learning ultrametric distances

by

Srinivas C. Turaga

Submitted to the Department of Brain & Cognitive Sciences
on August 14, 2009, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computation

Abstract

In this thesis I present new contributions to the fields of neuroscience and computer science. The neuroscientific contribution is a new technique for automatically reconstructing complete neural networks from densely stained 3d electron micrographs of brain tissue. The computer science contribution is a new machine learning method for image segmentation and the development of a new theory for supervised hierarchy learning based on ultrametric distance functions.

It is well-known that the connectivity of neural networks in the brain can have a dramatic influence on their computational function . However, our understanding of the complete connectivity of neural circuits has been quite impoverished due to our inability to image all the connections between all the neurons in biological network. Connectomics is an emerging field in neuroscience that aims to revolutionize our understanding of the function of neural circuits by imaging and reconstructing entire neural circuits.

In this thesis, I present an automated method for reconstructing neural circuitry from 3d electron micrographs of brain tissue. The cortical column, a basic unit of cortical micro-circuitry, will produce a single 3d electron micrograph measuring many 100s terabytes once imaged and contain neurites from well over 100,000 different neurons. It is estimated that tracing the neurites in such a volume by hand would take several thousand human years. Automated circuit tracing methods are thus crucial to the success of connectomics.

In computer vision, the circuit reconstruction problem of tracing neurites is known as image segmentation. Segmentation is a grouping problem where image pixels belonging to the same neurite are clustered together. While many algorithms for image segmentation exist, few have parameters that can be optimized using groundtruth data to extract maximum performance on a specialized dataset.

In this thesis, I present the first machine learning method to directly minimize an image segmentation error. It is based the theory of ultrametric distances and hierarchical clustering. Image segmentation is posed as the problem of learning and classifying ultrametric distances between image pixels. Ultrametric distances on point set have the special property that

2

they correspond exactly to hierarchical clustering of the set. This special property implies hierarchical clustering can be learned by directly learning ultrametric distances.

In this thesis, I develop convolutional networks as a machine learning architecture for image processing. I use this powerful pattern recognition architecture with many tens of thousands of free parameters for predicting affinity graphs and detecting object boundaries in images. When trained using ultrametric learning, the convolutional network based algorithm yields an extremely efficient linear-time segmentation algorithm.

In this thesis, I develop methods for assessing the quality of image segmentations produced by manual human efforts or by automated computer algorithms. These methods are crucial for comparing the performance of different segmentation methods and is used through out the thesis to demonstrate the quality of the reconstructions generated by the methods in this thesis.

Thesis Supervisor: H. Sebastian Seung
Title: Professor of Computational Neuroscience; Investigator, Howard Hughes Medical Institute

Contents

Bibliography   **100**

List of Figures

List of Algorithms

Chapter 1

Introduction

This thesis makes several contributions. Firstly, it presents an automated computational method for reconstructing neural circuitry from 3d electron micrographs solving a critical problem in the field of connectomics. Secondly, it presents the first method for directly training an image segmentation algorithm, developing a core method in machine learning and computer vision. Thirdly, it presents a general method for the supervised learning of ultrametric distance functions which is of general utility for the learning of hierarchical clustering. And finally (fourthly?), it introduces some novel measures for the quality of neurite tracings and image segmentation.

1.1 The dream of connectomics

The human brain has about 100 billion neurons, each of which connects to about 10 thousand neurons through chemical connections known as synapses. Far from being random, these connections between neurons are made in a highly structured way to form neural networks with distinct functions. While much is known about the gross anatomical wiring of the brain, little is known of the fine-scale local circuitry where most of the synapses are made.

A hypothesis is that the essential aspects of a local neural circuit are the list of all the neurons participating in the neural circuit, and the list of all the synapses between these neurons. We call this representation the *connectome*. It is hoped that the correlation between the structure of a neural circuit and its function will be strong enough for us to learn about
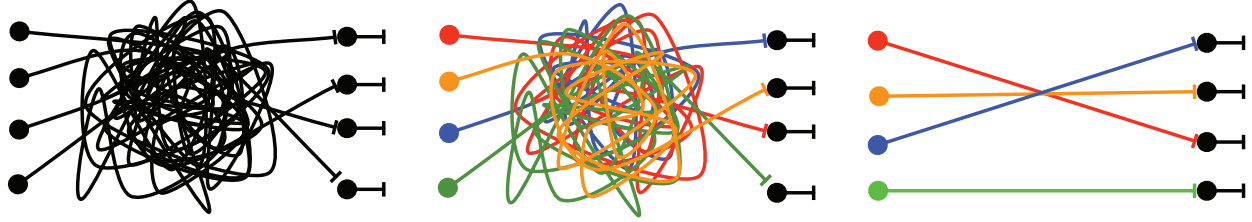
Figure 1-1: Neural circuit reconstruction requires the tracing of neurites through a tangled spaghetti of neuropil. Conceptually, the problem of tracing neurites is that of correctly assigning a unique color or identity to each neurite fragment. Such a coloring can then be used to generate a connectivity diagram.

the functioning of a neural circuit by studying its connectome.

In the last few years, technological advances have placed us closer to the ability to reconstruct the connectomes of neural circuits. Three dimensional electron microscopy has become feasible through the development of techniques such as Serial Block-Face Scanning Electron Microscopy (SBFSEM; Denk and Horstmann, 2004) and Automatic Tape-collecting Lathe Ultramicrotome (ATLUM; Hayworth et al. [2006]). These techniques are now poised to produce tens to hundreds of terabytes of imagery corresponding to volumes of brain tissue of no more than a fraction of a cubic millimeter.

1.2   The reconstruction problem

Unlike the regular organization in the circuitry of human-designed microprocessors, the circuitry of biological neural networks have little organization. The axons and dendrites constituting the wires of neurons form a dense tangled mess, akin to a bowl of branched spaghetti that has been compressed into a dense block. Reconstructing the connectivity of neurons is similar to figuring out which two connecting ends of cable in tangle of wires belong to the same cable. This process would be much simpler if the wires were color coded; this is the circuit reconstruction problem. Reconstructing neural circuits requires the tracing or "coloring" of neurite wires and synapses to make clear which synapses are formed between which two neurons.

The neurite tracing problem can be understood as a grouping or clustering problem. The task is to label all neurites belonging to the same neuron by the same color or unique identifier

and to label neurites belonging to different neurons with different colors. In computer vision, this grouping problem is known as image segmentation.

Image segmentation is defined as the partitioning of an image into distinct regions. There are many computer vision problems that either implicitly or explicitly perform image segmentation. For instance, a face detection algorithm built into an off-the-shelf digital camera finds image pixels that correspond to a face and implicitly segment an image into face pixels and non-face pixels. A more direct application of image segmentation involves the interpretation of a magnetic resonance images of the brain. In this case, a 3d image of the brain must be segmented into disjoint regions corresponding to different anatomical structures.

I 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 and Horstmann, 2004). This technique produces 3d images with a spatial resolution of roughly 30 nm or better. If applied to a tissue sample of just 0.5 mm x 0.5 mm x 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 non-neuronal 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 and Denk, 2006]. The sheer volume and complexity of data to be segmented precludes manual reconstruction and makes 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 et al., 1986, Fiala, 2005] to semi-automated interactive methods [Carlbom et al., 1994] and fully automated methods [Jurrus et al., 2006, Mishchenko, 2009]. 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 [Al-Kofahi et al., 2002, Jurrus et al., 2006, Andres et al., 2008, Mishchenko, 2009]. For

example, Jurrus et al. [2006] 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 semi-automated 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. [2008] use supervised learning in the form of a nearest neighbor classifier to segment 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].

1.3 The machine learning solution

Machine learning has been applied successfully to many computer vision problems. However, image segmentation has not benefited significantly from machine learning. This is for three reasons. (1) The lack of a trainable architecture for image segmentation. There has been some work on training classifiers for boundary detection [Fowlkes et al., 2003, Dollár et al., 2006], but the output of a boundary detector is not a final segmentation. A post-processing step such as a graph partitioning algorithm [Shi and Malik, 2000, Felzenszwalb and Huttenlocher, 2004] or watershed [Vincent and Soille, 1991] must be used to generate a segmentation, and there has not been a good way to optimize the parameters of these algorithms. (2) The lack of a standard metric for measuring segmentation performance. A machine learning based solution requires two components: a flexible trainable architecture, and a cost function by which to optimize performance. There has recently been some progress in this area. The work of Unnikrishnan et al. [2007] establishes the Rand Index as a good measure of segmentation performance. (3) The lack of a database with reliable groundtruth. The Berkeley Segmentation Data Set [Martin et al., 2001] has been an amazing resource for the computer vision community. It has found diverse applications from developing algorithms as diverse as image denoising and boundary detection. However, its use for benchmarking image segmentation algorithms has been limited due to the large variability

in the groundtruth segmentations of natural images. This large variability is useful in understanding the way humans process natural scenes, but detrimental to developing machine learning algorithms since the algorithm must model the variation as well as the average case. In this proposal, we make progress on all three of these issues.

The classic problems of regression and classification have been well-studied in the pattern recognition community resulting in the invention of neural networks, support vector machines and bayesian graphical models. Unfortunately, it is not immediately obvious how to apply these learning methods to the image segmentation problem. It has been recognized that the problems of image segmentation and clustering data points is closely related. This insight has led to the application of clustering algorithms such as k-means, mean shift and spectral methods to image segmentation [Hartigan and Wong, 1979, Comaniciu and Meer, 2002, Shi and Malik, 2000].

Clustering-based algorithms usually 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. The 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 and Malik, 2000, Fowlkes et al., 2003].

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 datasets. 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]. In part, this may have been for lack of a good image processing architecture that can discover the appropriate features directly from the raw image.

Here, I propose a novel machine learning architecture that unifies clustering with classification. The *connectivity classifier* for learning clustering.

The research proposed in this thesis is about the development of a new method. The novel parts of this method will be described in the next section. Here, I describe general

methodologies that are common to all machine learning algorithms.

1.3.1 Groundtruth datasets

A machine learning algorithm requires data to learn from. For learning of image segmentation, we will require a groundtruth dataset in the form of raw imagery and corresponding human expert generated segmentations.

Given our focus on developing tools for connectomics, I shall use electron microscopic imagery acquired through a collaboration with the laboratory of Winfried Denk of the Max-Planck Institute of Heidelberg, Germany. These images are generated by imaging with the SBFSEM technique [Denk and Horstmann, 2004]. The dataset will correspond to several 3d images acquired at a resolution of approximately 25nm x 25nm x 25nm. These images have been segmented by human experts at the Denk laboratory.

There exist a number of databases of human segmented natural images. For example, the Berkeley Segmentation Data Set (BSDS, Martin et al. [2001]) and the LabelMe dataset. However, in contrast to the human segmentations of EM images, natural images can have extremely variable interpretations and context dependent segmentations. I believe that it would be easier to develop a machine learning algorithm using the more reliable EM database.

1.3.2 Validation

Machine learning is the art of creating an algorithm that once trained on one dataset will generalize to other datasets. The most worrisome pitfall of this strategy is that of overfitting. This is where the machine learns to perform really well on the training dataset, but has poor performance on other datasets.

A common strategy for detecting this problem is to use two disjoint groundtruth datasets. The first *training* dataset is used to train the machine, and the generalization of this machine is tested using the *test* dataset. A good machine will have equally good performance on both the training and test datasets.

More complicated versions of this validation strategy exist, such as leave-one-out cross validation. We will use these as needed.

1.4 Thesis organization

This thesis introduces a formalism for generating hierarchical segmentations of images based in large part on the minimum spanning tree representation. This constitutes a unification of existing segmentation algorithms based on boundary maps and affinity graphs. I present this formalism in Chapter 2.

This thesis develops the use of a machine learning architecture for image processing known as a convolutional network. These networks can be used as boundary detectors and for affinity graph generation. I present the basics of convolutional network architecture and training procedure in Chapter 4 and demonstrate how to apply these networks to generate affinity graphs in Chapter 5.

This thesis presents the first truly end-to-end segmentation learning algorithm. Based on the theory of hierarchical partitioning and ultrametric distance functions, I develop a new learning algorithm for training a machine learning architecture to produce hierarchical segmentations that directly minimize a measure of segmentation error, rather than proxy error measures as in the past. Chapter 6 presents the theory of ultrametric distances, their relationship to hierarchical clustering and gives a new learning algorithm for learning to generate hierarchies. Chapter 7 presents a specialization of the minimax ultrametric learning algorithm to learning image segmentation by a convolutional network for affinity graphs.

And finally Chapter 8 discusses the relationship of the work presented here to prior work in image segmentation and clustering. We conclude with some directions for future research.

Chapter 2

Hierarchical segmentations of boundary maps and affinity graphs with minimum spanning trees

An *image segmentation* is a partitioning of image pixels into regions corresponding to different objects. A *hierarchical image segmentation* is a recursive grouping of image regions or pixels. Regions of high confidence can be merged together to form larger regions of lower confidence. Such a representation is also useful when an image can be segmented into objects which themselves can be further sub-divided into parts of objects with specific meaning. This chapter describes a general means of generating hierarchical segmentations by means of a minimum spanning tree computation on a graph over image pixels.

There are two classic approaches to generating hierarchical segmentations based on the different representations of boundary maps and affinity graphs. I will show how both apparently disparate approaches can be unified in the context of computing minimum spanning trees. Hierarchical segmentations generated in this way are special in that they define a particular distance over image pixels known as an ultrametric distance. We will use this fact in Chapter 6 to develop a learning algorithm for training boundary detectors and affinity functions. Together, this chapter and Chapter 6 constitute a new understanding of classic image segmentation algorithms, and the first machine learning algorithm for image segmentation to directly optimize a segmentation error. The resulting algorithm is presented in

Chapter 7.

The algorithms for generating hierarchical segmentations generalize Kruskal's minimum spanning tree and the connected components algorithms [Cormen et al., 1990] for graphs to segment affinity graphs and boundary maps. These generalizations include the recent efficient affinity graph partitioning algorithm by Felzenszwalb and Huttenlocher [2004], and such classic boundary map segmentation algorithms as the watershed transform [Vincent and Soille, 1991, Meyer, 1994], Ultrametric Contour Maps [Arbelaez, 2006] and the image connected components labeling algorithm [Gonzalez and Woods, 2002]. An important goal of this chapter is present a unified view of these disparate algorithms for the first time.

I will start by reviewing the basic graph theory that will be useful for the discussion.

2.1 Graph representations and algorithms

A graph G is defined by a set of vertices or nodes V and a set of edges E that connect the vertices. Edges and nodes can be weighted or not. Edges can be directed or not. In this thesis, we will use graph representations in two very distinct ways.

Note: we will use undirected, edge-weighted graphs to represent segmentations. And we will use directed graphs to represent convolutional networks in Chapter 4. This chapter is concerned with the former.

2.1.1 Minimum spanning trees

A spanning tree of a graph G is a graph T that contains all the vertices of G and a subset of the edges in G that *span* all the vertices. A minimum spanning tree T_{MST} of a weighted graph G is the subset of edges chosen such that they minimize the sum of the edge weights.

A spanning tree has the property that it contains no loops. This property can be leveraged to construct a greedy algorithm for finding MSTs. Kruskal's MST algorithm is one such algorithm. Kruskal's algorithm constructs an MST by starting with a graph with all the original vertices and none of the edges. Edges are then added to this graph one at a time in increasing order of weight so long as they do not create a cycle.

The main computational step in Kruskal's algorithm is determining whether an edge
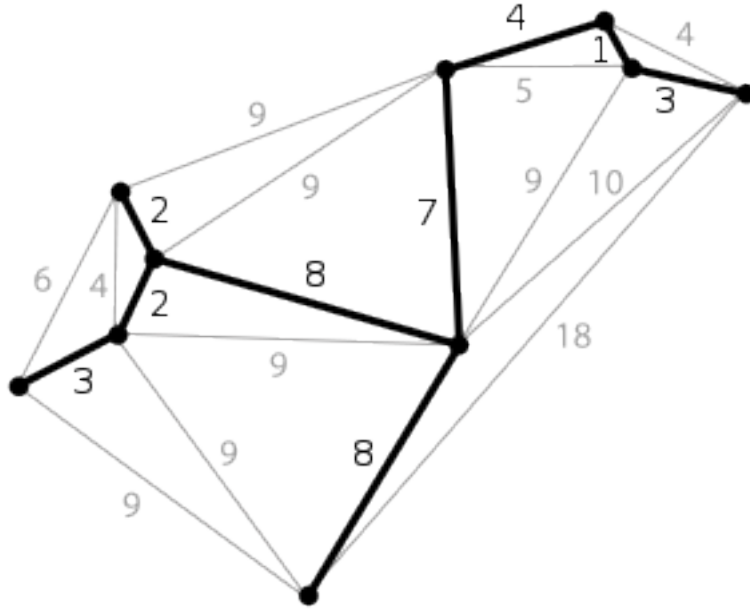
Figure 2-1: The minimum spanning tree (black edges) of an undirected weighted graph (all edges). Note that the spanning tree has no loops, and that it connects all graph vertices.

would create a cycle if added. This is done by maintaining and updating a "clustering" of all the vertices in the graph. Initially, all the vertices start out in separate clusters of size one. Edges are only added to the tree if each of the two vertices connected by the edge belong to two different clusters. And adding an edge leads to the merging of these two clusters. This construction guarantees that clusters contain vertices that are connected to each other by paths. Thus edges whose vertices are in the same cluster are never added since they will always create loops.

An efficient implementation of Kruskal's algorithm using a *soft heap* and *disjoint sets* data structures [Chazelle, 2000, Cormen et al., 1990] has a runtime complexity of $O(|E| \cdot \alpha(|E|))$, where $|E|$ is the number of edges in the graph and $\alpha(\cdot)$ is the inverse Ackermann function that grows *extremely* slowly. In fact, $\alpha(n)$ is less than 5 for any conceivable input, since $\alpha(2^{2^{10^{19729}}}) = 4$. Thus Kruskal's MST algorithm is roughly linear in the number of edges of the graph.
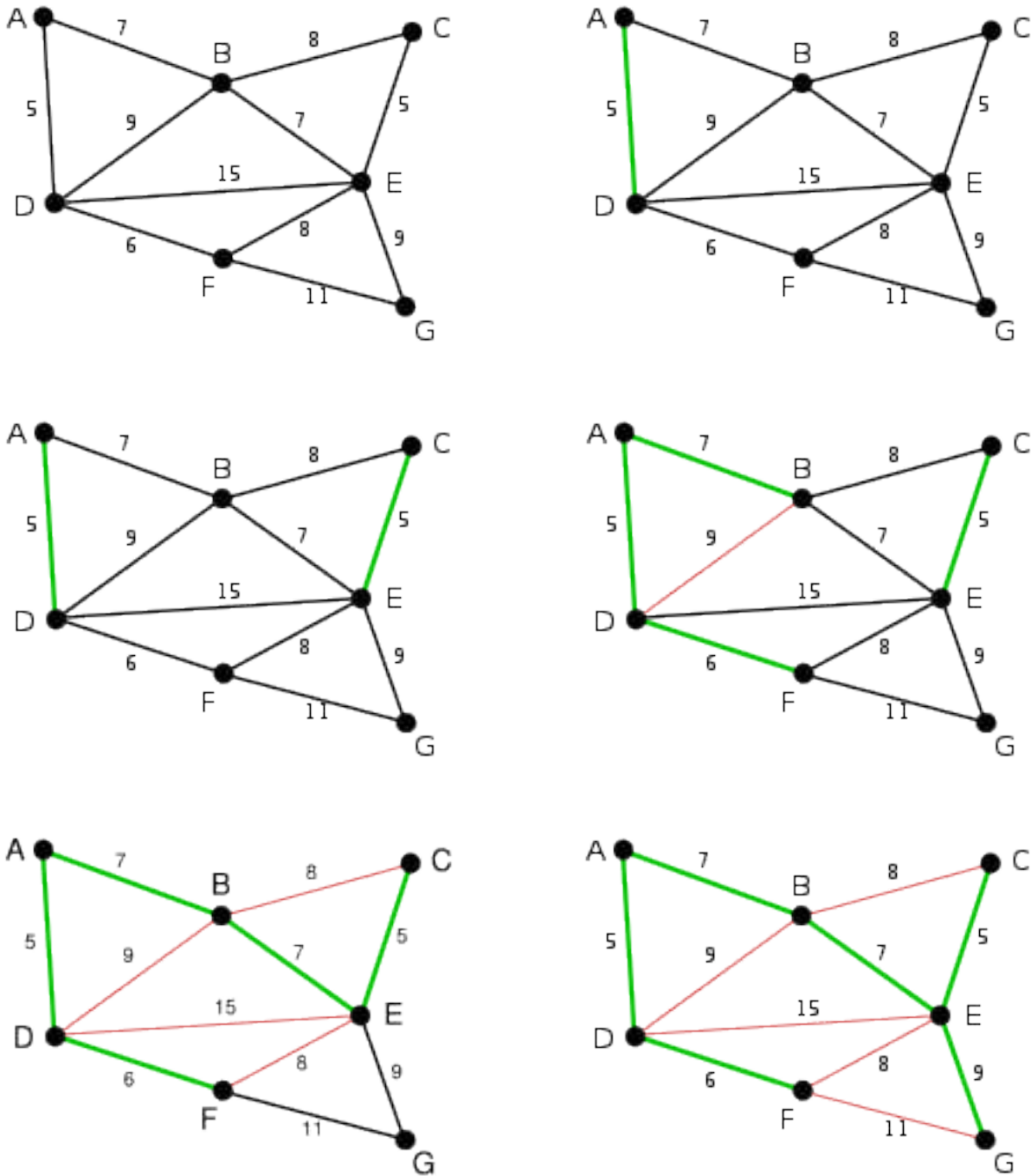
Figure 2-2: **Stages of Kruskal's minimum spanning tree algorithm.** (Left to right, top to bottom) Edges are examined in increasing order of weight, and are added (green) to the MST if they do not create a loop. Loop-causing edges are ignored (red). In the case of edges of equal weight, ties are broken arbitrarily.

Algorithm 2.1 KruskalMST(V,E)

Initialize graph T with the vertices V and no edges

Initialize clustering of vertices with each vertex in its own cluster

For each edge E_{ij} in increasing order of weight, breaking ties arbitrarily

    if IN-SAME-CLUSTER(i,j)

        then do nothing
        else MERGE(CLUSTER(i),CLUSTER(j)), add E_{ij} to T

Return T

2.1.2   Hierarchical clustering, dendrograms and MST

Kruskal's algorithm can be viewed as an algorithm that starts out with a clustering of all the vertices in the graph into separate clusters and then iteratively updates the clustering. However, our updating of the clusters is quite constrained. We only ever merge pairs of clusters; we never split clusters or reassign individual vertices. This iterative merging algorithm corresponds to a class of clustering procedures known as hierarchical clustering. In particular, the MST corresponds to single-linkage hierarchical clustering.

Hierachical clusterings can be visualized using dendrograms. A dendrogram is a rooted binary tree whose leaf nodes consist of the objects being clustered. In our case, this corresponds to the vertices of the graph. Each internal node of this tree represents a cluster corresponding to all its child leaf nodes. In the case where a hierarchical clustering of nodes is generated by the minimum spanning tree of a weighted graph, the internal nodes also have a one-to-one correspondence with edges in the MST, and the height of each internal node represents the weight of its corresponding edge.

2.1.3   Connected components and single-level clustering

A single clustering of the vertices of a graph can be generated from a hierarchical clustering by "cutting" the dendrogram at a chosen height threshold. This corresponds to pruning away all edges in the MST of weight greater than the threshold. The connected components of this pruned graph constitute a clusters.
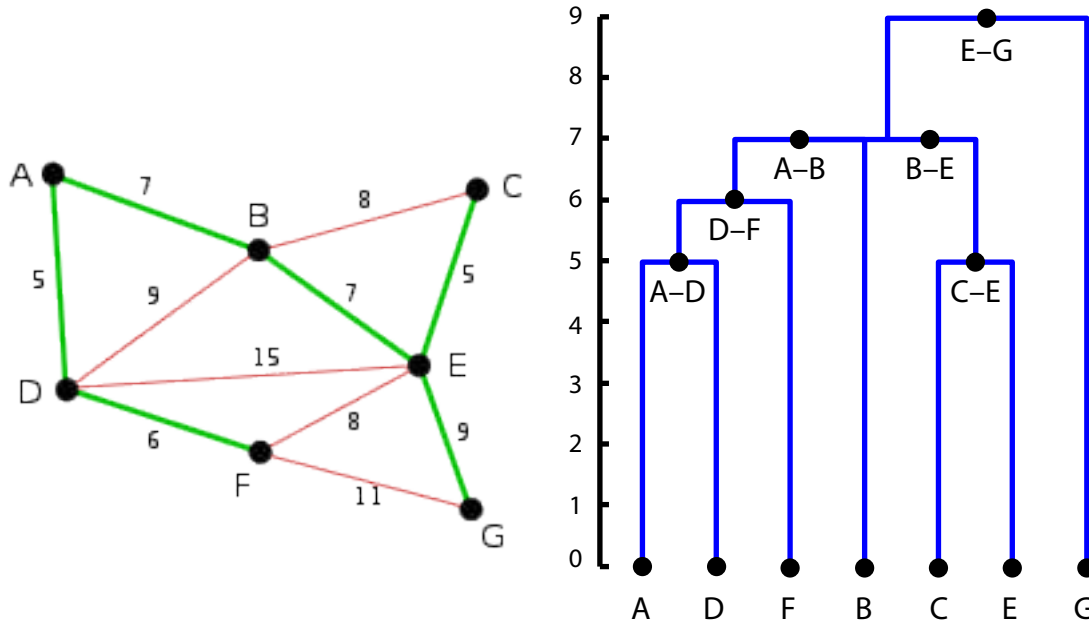
Figure 2-3: A dendrogram representation of a minimum spanning tree. The leaf nodes in the dendrogram correspond to vertices in the MST. The internal nodes in the dendrogram correspond to edges in the MST. The height of the internal nodes corresponds to the weight of the edge in the MST.

The same single clustering can be generated in a more efficient manner by directly pruning the edges in the original graph that are greater than the threshold and finding the connected components of this pruned graph. Here, we exploit the fact that the connected components of the pruned original graph are identical to the connected components of the pruned MST. This fact can be seen by noticing that extra edges in the pruned graph as compared to the pruned MST do not impact the connectivity of the connected components.

2.2 Hierarchical segmentations of affinity graphs

2.2.1 Affinity graphs and graph partitioning

An *affinity graph* is a weighted graph A whose nodes correspond to image pixels and edges A_{ij} between image pixels i and j reflect the affinity of the pixel pair (i, j) to be grouped together. In an affinity graph, edges with large weights between pixels are interpreted to mean that the pixels belong to the same segment and vice versa.

A segmentation algorithm involving the use of an affinity graph has two stages. The first computation is that of the prediction of an affinity graph from the image. This is generally viewed as a low-level vision operation involving the detection of local boundary cues such as as edges defined by image gradients. The second computation involves "cutting" or "partitioning" the affinity graph to generate an image segmentation. This is generally viewed as a global integration of local cues based on a prior assumption about what constitutes good segmentations.

The affinity between two pixels is frequently estimated by means of a hand designed function that compares the similarity of the two image patches containing the pixels. Commonly used features for comparison include pixel intensities, local color histograms and local texture histograms [Shi and Malik, 2000, Felzenszwalb and Huttenlocher, 2004]. Commonly used graph partitioning algorithms include Normalized Cuts [Shi and Malik, 2000], Graph Cuts [Boykov et al., 2001] and the MST-based "efficient graph partitioning" algorithm by Felzenszwalb and Huttenlocher [2004].

Recently, there has been some work on using supervised machine learning to predict the weights of edges in the affinity graph. In principle, one would like to train the affinity prediction module to produce affinity graphs which once partitioned, would yield good segmentations. However, this is a non-trivial machine learning problem [Bach and Jordan, 2006, Cour et al., 2005b, Meila and Shi, 2001]. It involves understanding exactly how small changes in the weight of individual affinity edges leads to changes in image segmentation in order to compute a gradient of a segmentation error measure with respect to the weight of affinity edges. Approximate solutions to this problem for the normalized cuts graph partitioning algorithm have been suggested, but they suffer from significant technical problems, many of which are inherent to the normalized cuts algorithm [Bach and Jordan, 2006, Cour et al., 2005b, Meila and Shi, 2001].

In practice, the supervised learning of affinity graphs has involved training the prediction module to minimize a proxy purely local, non-segmentation error measure [Fowlkes et al., 2003, Turaga et al., in press]. We demonstrate this approach for the convolutional network architecture for producing affinity graphs in Chapter 5. In Chapter 7, we introduce an algorithm based on ultrametric learning, that for the first time allows the exact training of

the same convolutional network architecture to minimize a segmentation error [Turaga et al., submitted].

2.2.2 Graph partitioning and hierarchical segmentation by maximum spanning trees

In this section, I propose an algorithm for generating hierarchical image segmentations using spanning trees over image pixels. This algorithm is closely related to the "efficient graph partitioning algorithm" by Felzenszwalb and Huttenlocher [2004]. The idea is that an image segmentation corresponds to a clustering of image pixels. Thus, a spanning tree computed from a weighted graph over image pixels can be used to represent a hierarchical image segmentation by clustering the pixels. Individual segmentations can be generated by simply finding the connected components of the weighted graph with edges of affinity lower than a given threshold pruned.

Affinity graphs have the opposite semantics of the general graphs we discussed earlier. High edge weight between a pixel pair reflects high affinity for them to be part of the same segment. For this reason, we propose generating hierarchical segmentations using the *maximum spanning tree* algorithm. We define the *maximum spanning tree* as the spanning tree with the largest edge weights. This is computed in exactly the same manner as Kruskal's MST algorithm, except that we sort the edges in decreasing order of cost. This yields a hierarchical segmentation algorithm that is roughly linear in the number of edges in the graph, since the maximum spanning tree algorithm has complexity $O(|E| \cdot \alpha(|E|))$. For images that contain *simply-connected* objects, a nearest-neighbor affinity graph is sufficient to encode all possible segmentations. Such a graph has $|E| = k|V|$ for some constant k. But if the images contain occluded objects, where there may not be a simply-connected path of pixels between the two regions of an image, an all-to-all connected graph is necessary to represent all possible segmentations, requiring a weighted graph with $|E| = |V|^2$.

An image segmentation at a given height of the hierarchical segmentation dendrogram can be computed even more efficiently than the full hierarchical segmentation, which requires the spanning tree. This image segmentation corresponds to the connected components of
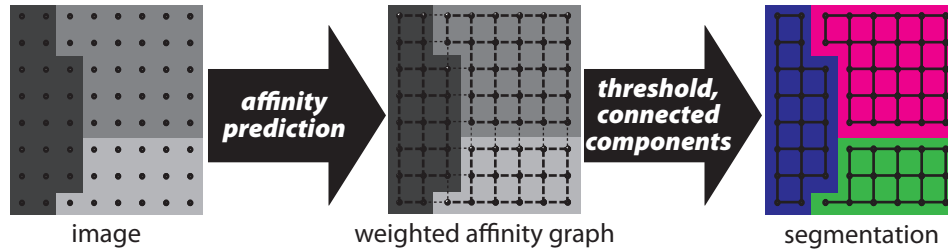
Figure 2-4: **Segmenting with affinity graphs.** Nearest neighbor affinity graphs encode hierarchical segmentations. A single level of the image segmentation can be computed efficiently by edge thresholding, followed by connected components.

the affinity graph with edges of affinity less than given threshold pruned away.

2.3   Hierarchical segmentations of boundary maps

A boundary map is an image where the value B_i of pixel i represents the degree to which pixel i constitutes a boundary. Classically, high values are used to represent boundary pixels and low values represent the absence of an object boundary as shown in Fig. 2-5. However, SBFSEM images naturally look like noisy boundary maps, with boundaries having low image intensities[1]. So we adopt an inverted representation for SBFSEM boundary maps where boundaries are low and the interiors of objects as high; a representation we sometimes refer to as in-out for "in"-side of cells and "out"-side of cells (boundaries). An example of this may be seen in Fig. 2-6.

A binary boundary map can be regarded as representing a segmentation in the sense that image regions bounded by boundary pixels correspond to segments. The image connected components labeling algorithm provides a means of performing this operation in $O(N)$ for an image with N pixels.

---

[1]Interestingly, the raw image generated from scanning backscattered scanning electron microscopy techniques such as SBFSEM look much like the boundary maps in Fig. 2-5 with high backscatter intensities near electron dense boundaries. However, by convention, SBFSEM images are inverted to match transmission electron microscopy images.
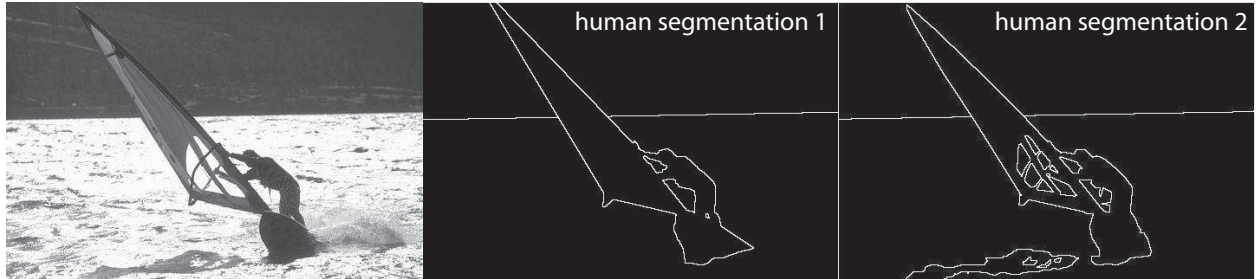
Figure 2-5: Binary boundary maps generated from human segmentations of natural images from the Berkeley Segmentation Data Set [Martin et al., 2001]. Here, boundaries are represented as high with 1s and non-boundaries as low with 0s.
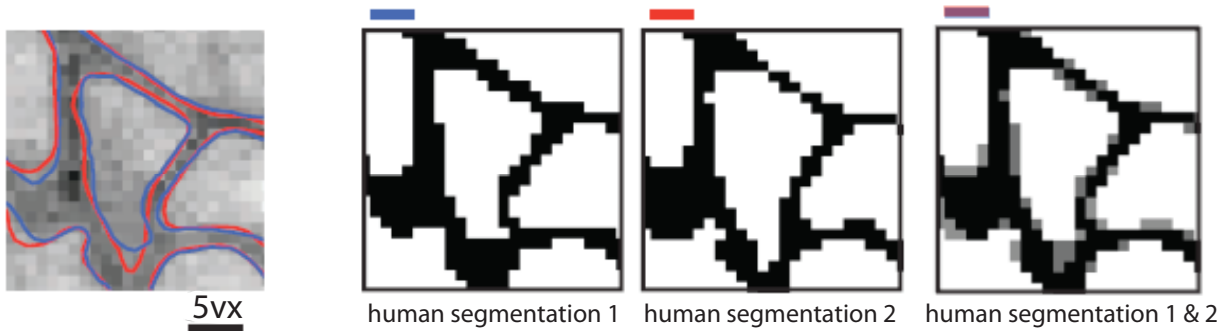
Figure 2-6: Binary "in-out" boundary maps generated from human segmentations of SBF-SEM images. In contrast to the previous figure, boundaries are represented as low with 0s and non-boundaries are high. This inversion of representation is of no real consequence, but corresponds well to our intuitive notion of a "denoised" binary SBFSEM image, since the original SBFSEM images have the same semantics.

SBFSEM image   in-out boundary map   height map interpretation
                                                    of boundary map
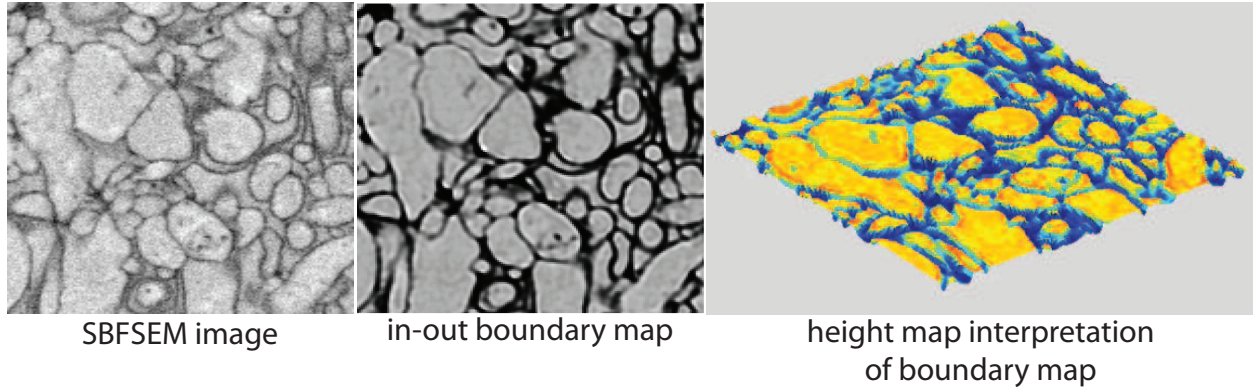
Figure 2-7: Continuous boundary maps and flooding. Raising and dropping the "water-level" changes the segmentation by splitting and merging segments.

2.3.1 Continuous boundary maps and flooding

Binary boundary maps have a unique segmentation interpretation when using connected components. However, a continuous boundary map is a richer representation in that it can be regarded as representing a hierarchy of nested segmentations.

One way to understand this is to consider thresholding the boundary map to generate binary boundary maps at different thresholds. Connected components labeling of each of these thresholded boundary maps will lead to different segmentations.

Suprisingly, the hierarchical segmentations generated by flooding correspond exactly to performing a maximum spanning tree computation on a weighted nearest-neighbor affinity graph generated from the boundary map as $A_{ij} = \min(B_i, B_j)$ for nearest neighbor pixels i, j according to the definition of topological connectedness used for the flooding.

2.4 Comparing affinity graph and boundary map representations

While both affinity graphs and boundary maps can represent hierarchical segmentations, there is a difference in the representational power of these representations. In particular, the boundary map can only represent simply-connected segments; occluded objects that are disconnected in pixel space cannot by segmented as one object. The nearest-neighbor affinity graph suffers from the same problem. An all-to-all connected affinity graph is required in

32

A. Segmentations resulting from changing the "flooding" threshold

B. Dendrogram representing the hierarchy
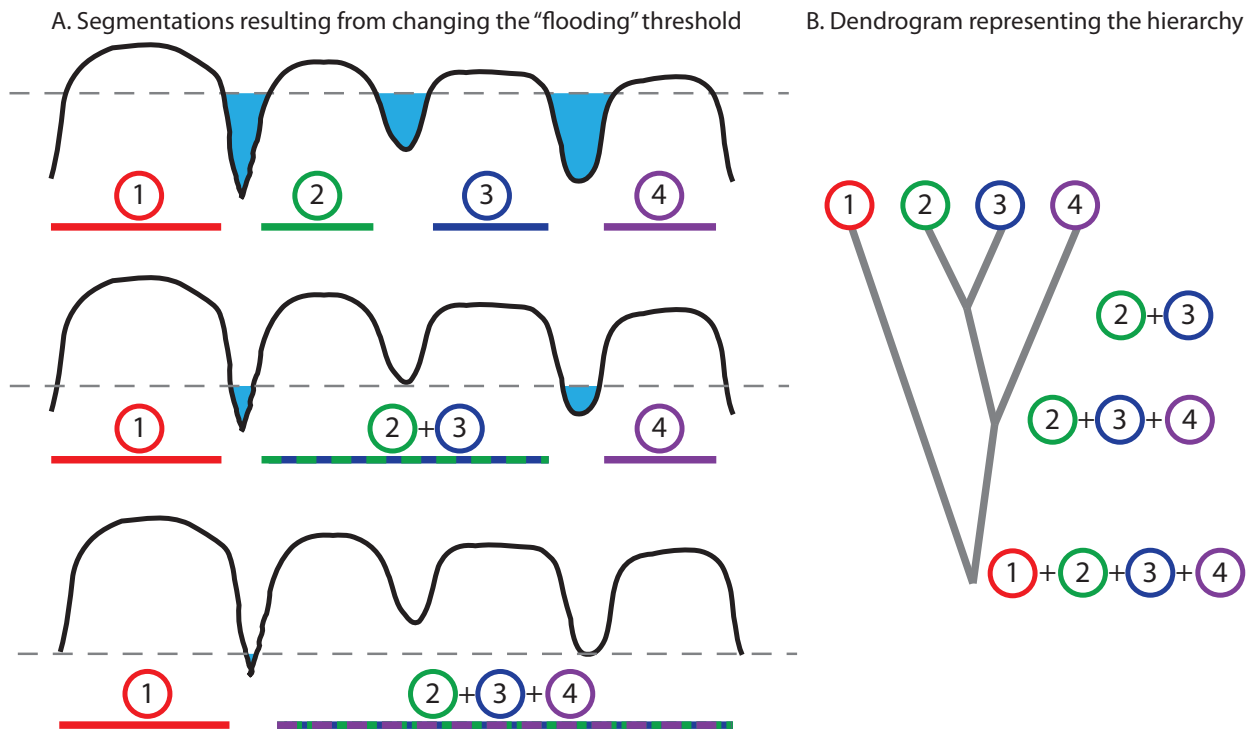
Figure 2-8: Hierarchical segmentation of a hypothetical 1d in-out boundary map. "Flooding" by varying the threshold (gray dashed line) for binarizing the continuous boundary map. A region merging hierarchy is formed by lowering the threshold.

these situations.

In 3d serial block-face scanning electron microscopic images of neural tissue, there is no occlusion, and a nearest neighbor affinity graph is sufficient to encode all image segmentations, but a boundary map might be insufficient. However, due to the limited imaging resolution, the neighborhood required might be a radius as large as several pixels large.

In other serial electron microscopy methods, there is frequently a large difference in the $x - y$ vs. z resolution, and subsequent image sections may be imperfectly aligned. A boundary map representation is insufficient to represent image segmentations in these images. A nearest neighbor affinity graph with a small same-section $x - y$ radius, but large across-section $x - y$ radius is required to represent all possible segmentations.

Chapter 3

Measures of reconstruction quality

We will want to measure the quality of reconstructions for three reasons.

Absolute performance. How well are we doing? How many mistakes are in a final reconstruction? How much time will it take a human expert to proof-read and correct a machine reconstruction? What is the error rate inherent to a reconstruction technique?

Relative performance. Which method for producing segmentations is better? By how much? Which human experts are more reliable than others?

Optimizing performance. Training machine learning algorithms. Automatically adjusting the parameters of an algorithm to improve segmentation performance. Generalization performance of a trained algorithm.

In contrast to classification and regression where well-defined performance measures exist, there are few good measures for quantifying segmentation and circuit reconstruction performance. We have developed a few such measures, some of which have been generalized from the classification literature, which we detail in the following section.

Up until this point, we have been treating the problems of neurite tracing and image segmentation as identical. But there is a subtle way in which the segmentation of one dimensional cylinder-like objects is different from generic image segmentation. In tracing neurons, the primary goal is to ensure that all the synapses are correctly grouped with the same cell body. For this, we care more deeply about the correct grouping of image pixels

along the neurite, but not too much about getting the thickness of the neurite exactly right. In this sense, it is sufficient to correctly segment the *skeleton* of the neurite.

Pixel-to-pixel measures compare two segmentations that have been generated at the finest pixel scale and are useful for comparing generic image segmentations. A primary consideration here is to correctly assign the connectivity of the segmentations

Pixel-to-skeleton measures can be used to compare pixel-level image segmentations to skeletonized image segmentations and are preliminarily useful for segmentations of generic cylinder-like structures including neurons. Here too, we will be primarily concerned with the connectivity.

Skeleton-to-skeleton measures will be used to compare two skeletons for segmentations of branching cylindrical neuron-like tree structures. In this case, we will be interested in comparing the topologies of the reconstructed skeletons.

3.1 Basic classic performance measure primitives and their interpretation

3.1.1 Binary classification error

For measuring the performance of a predictor of a binary value, the simplest quantity is the misclassification rate. $E_{\text{binary}} = \frac{1}{N} \sum_i \delta(\hat{y}_i, y_i)$ where $\delta(x, y)$ is 1 if $x = y$ and 0 otherwise. This metric lumps together two kinds of errors which can be useful to measure separately.

False positives are errors of commission. These are examples that are in truth negative, but erroneously declared positive by the predictor.

False negatives are errors of omission. These are examples that are in truth positive, but erroneously declared negative by the predictor.

To see the need for explicitly measuring false positives and negatives, consider the scenario where the training dataset contains a large imbalance in the number of true positive and

negative examples. For instance, a bomb detector at an airport may contain a few examples of actual bombs (let's say 1%), but many many times more examples of situations where no bombs were encountered (99%), reflecting the normal statistics of operation. (Even a 1% rate of encountering bombs is probably abnormally high. One would not want to use this airport.) A trivially "good" predictor that always reports a negative bomb sighting would be considered bad, even though it is right 99% of the time and would have an $E_{\text{binary}} = 0.01$. We would only understand why this is bad when we notice that our predictor has a bias problem – it only makes false negative errors but no false positives. Further, it declares *every* positive example falsely negative! (One would *never* want to use an airport with such a bomb detector.)

There are a few established ways of summarizing false positive and false negative information which I review here.

3.1.2   Confusion matrices

A confusion matrix is a visual representation of the performance of a binary classifier. A perfect classifier would have zero false negatives and positives and hence have a confusion matrix that has no off diagonal entries. The larger the off-diagonal entries, the worse the classifier.

<div align="center">

actual value

		p	n	**total**
prediction	$\hat{p}$	True positive (TP)	False positive (FP)	$\hat{P}$
outcome	$\hat{n}$	False negative (FN)	True negative (TN)	$\hat{N}$
	total	P	N	

</div>

3.1.3   Receiver Operating Characteristic (ROC) for binary classification

In signal detection theory, which relates to such things as the detection of bombs, a *reciever operating characteristic* is a graphical plot of the *true positive rate (TPR)* vs. the *false positive rate (FPR)* of a detector/classifier/predictor, where $TPR = TP/P$ and $FPR = FP/N$. A good classifier would have a very high TPR (close to 1) and a very low FPR (close

to 0).

While a given binary classifier would only have a single (TPR, FPR) corresponding to a single point on a ROC plot, ROC curves can be generated for parametrized classifiers by scanning the parameters. For binary predictors that produce an analog confidence of the binary prediction instead of just a binary output, this confidence value must be thresholded to yield a binary prediction. Thus, many different binary classifiers can be created, based on what value of threshold is chosen. The ROC curve is generated by computing TPR and FPR for different values of the threshold.

3.1.4   Precision-recall for binary classification

In information retrieval theory, which relates to such things as the search engines and databases, a *precision-recall curve* measures the number of items that are returned by a search when compared with the number of actually relevant items.

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 $Precision = TP/(TP + FP)$, while recall measures the total fraction of positive examples that are classified positive $Recall = TP/(TP + FN)$. 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.

3.1.5   Rand index of clustering performance

In clustering theory, the Rand index [Rand, 1971] is a measure of similarity between two partitionings of a dataset. A clustering of N items into k clusters can be represented as a labeling of each item by an integer in $(1, 2...k)$ such that items belonging to the same cluster

share the same label S_i.

The Rand index measures the number of pairs of points in the data set that are correctly grouped together. Defining the "same cluster" matrix SC for each clustering as $SC_{ij} = \delta(S_i, S_j)$, we can define the Rand index as measuring the discrepancy between the "same cluster" matrices for two different clusterings.

$$RI(SC^1, SC^2) = \binom{N}{2} \sum_i \sum_{j<i} \delta(SC_{ij}^1, SC_{ij}^2)$$

3.2 Boundary error

An image segmentation can be used to generate a boundary map over the image. A simplistic measure of image segmentation is to simply quantify the number of pixels incorrectly classified as boundary or not.

$$E_{boundary} = \frac{1}{N} \sum_i \left| B_i - \hat{B}_i \right|$$

This measure has the nice property that a boundary detector can be easily be trained to minimize this boundary misclassification error.

Unfortunately, this measure is only partly reflective of segmentation performance. Firstly, boundaries can frequently be displaced by small amounts leading to large boundary misclassification errors, but leave the segmentation unchanged. This is most commonly a problem with human generated segmentations where humans frequently draw boundaries that can be quite variable

The biggest problem with this measure is that it has little to do with segmentation performance. A boundary detector need only make a few mistakes to dramatically alter the segmentation interpretation of the boundary map.
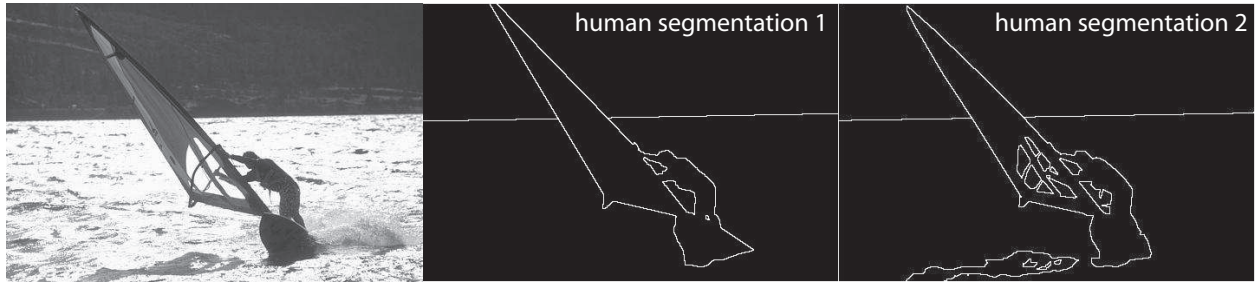
Figure 3-1: Human segmentations of natural images exhibit a large degree of variability. Both boundary locations as well as the granularity of segmentation can be different. Adapted from [Martin et al., 2004].
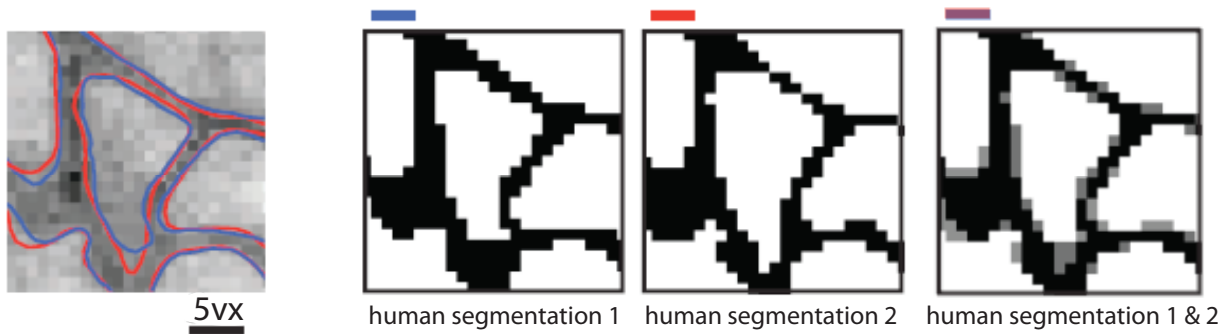
Figure 3-2: Boundaries in human generated image segmentations of EM images have large variability but there are few topological disagreements.
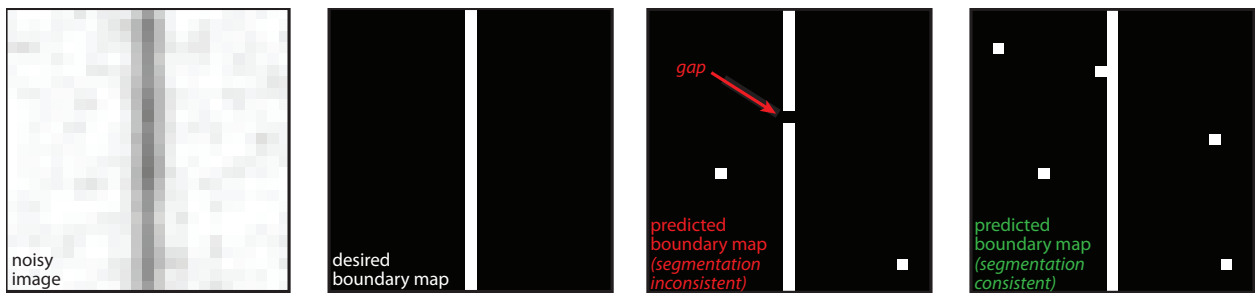
Figure 3-3: Boundary classification is a poor measure of segmentation performance.

3.3 Segmentation error

3.3.1 The Rand index quantifies segmentation performance

Image segmentation can be viewed as a special case of the general problem of clustering, as image segments are clusters of image pixels. Long ago, Rand proposed an index of similarity between two clusterings Rand [1971]. Recently it has been proposed that the Rand index be applied to image segmentations [Unnikrishnan et al., 2007]. Define a segmentation S as an assignment of a segment label s_i to each pixel i. The indicator function $\delta(s_i, s_j)$ is 1 if pixels i and j belong to the same segment $(s_i = s_j)$ and 0 otherwise. Given two segmentations S and $\hat{S}$ of an image with N pixels, define the function

$$1 - RI(\hat{S}, S) = \binom{N}{2}^{-1} \sum_{i<j} |\delta(s_i, s_j) - \delta(\hat{s}_i, \hat{s}_j)| \tag{3.1}$$

which is the fraction of image pixel pairs on which the two segmentations *disagree*. We will refer to the function $1 - RI(\hat{S}, S)$ as the Rand index, although strictly speaking the Rand index is $RI(\hat{S}, S)$, the fraction of image pixel pairs on which the two segmentations *agree*. In other words, the Rand index is a measure of similarity, but we will often apply that term to a measure of dissimilarity.

In this paper, the Rand index is applied to compare the output $\hat{S}$ of a segmentation algorithm with a ground truth segmentation S, and will serve as an objective function for learning. Figure 7-1 illustrates why the Rand index is a sensible measure of segmentation performance. The segmentation of affinity graph #1 incurs a huge Rand index penalty relative to the ground truth. A single wrongly classified edge of the affinity graph leads to an incorrect merger of two segments, causing many pairs of image pixels to be wrongly assigned to the same segment. On the other hand, the segmentation corresponding to affinity graph #2 has a perfect Rand index, even though there are misclassifications in the affinity graph. In short, the Rand index makes sense because it strongly penalizes errors in the affinity graph that lead to split and merger errors.
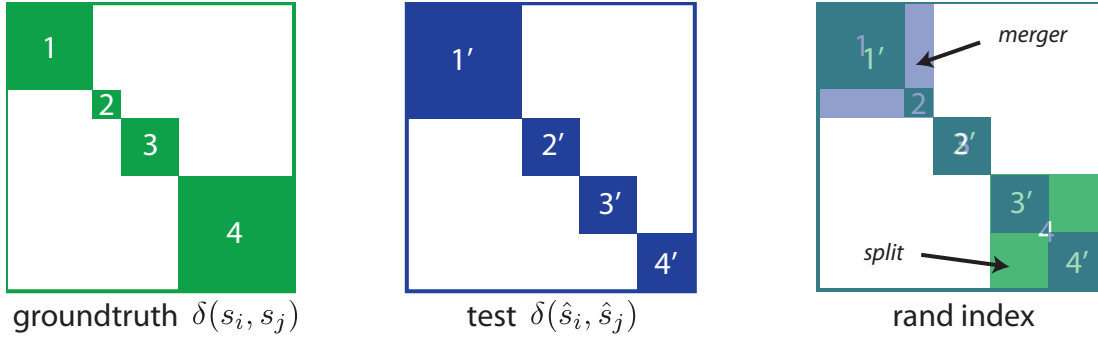
Figure 3-4: **The Rand index quantifies segmentation performance by comparing the difference in pixel pair connectivity between the groundtruth and test segmentations.** Pixel pair connectivities can be visualized as symmetric binary block-diagonal matrices $\delta(s_i, s_j)$. Each diagonal block corresponds to connected pixel pairs belonging to one of the image segments. The Rand index incurs penalties when pixels pairs that must not be connected are connected or vice versa. This corresponds to locations where the two matrices disagree. An erroneous merger of two groundtruth segments incurs a penalty proportional to the product of the sizes of the two segments. Split errors are similarly penalized.

3.3.2   Measuring large-scale connectivity errors: splits and mergers

The number of splits and mergers caused by an algorithm is a useful measure of segmentation performance. Split errors are quantified as the number of fragments an object in the ground truth dataset 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 groundtruth 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 groundtruth 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 groundtruth 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 groundtruth is erroneously broken into

a) groundtruth segmentation
no splits or mergers

b) 1 merger

test
segmentation
(overlay)

groundtruth
segmentation
(underlay)

c) 1 split

d) 2 splits + 1 merger

objects in
groundtruth
segmentation

objects in
test
segmentation
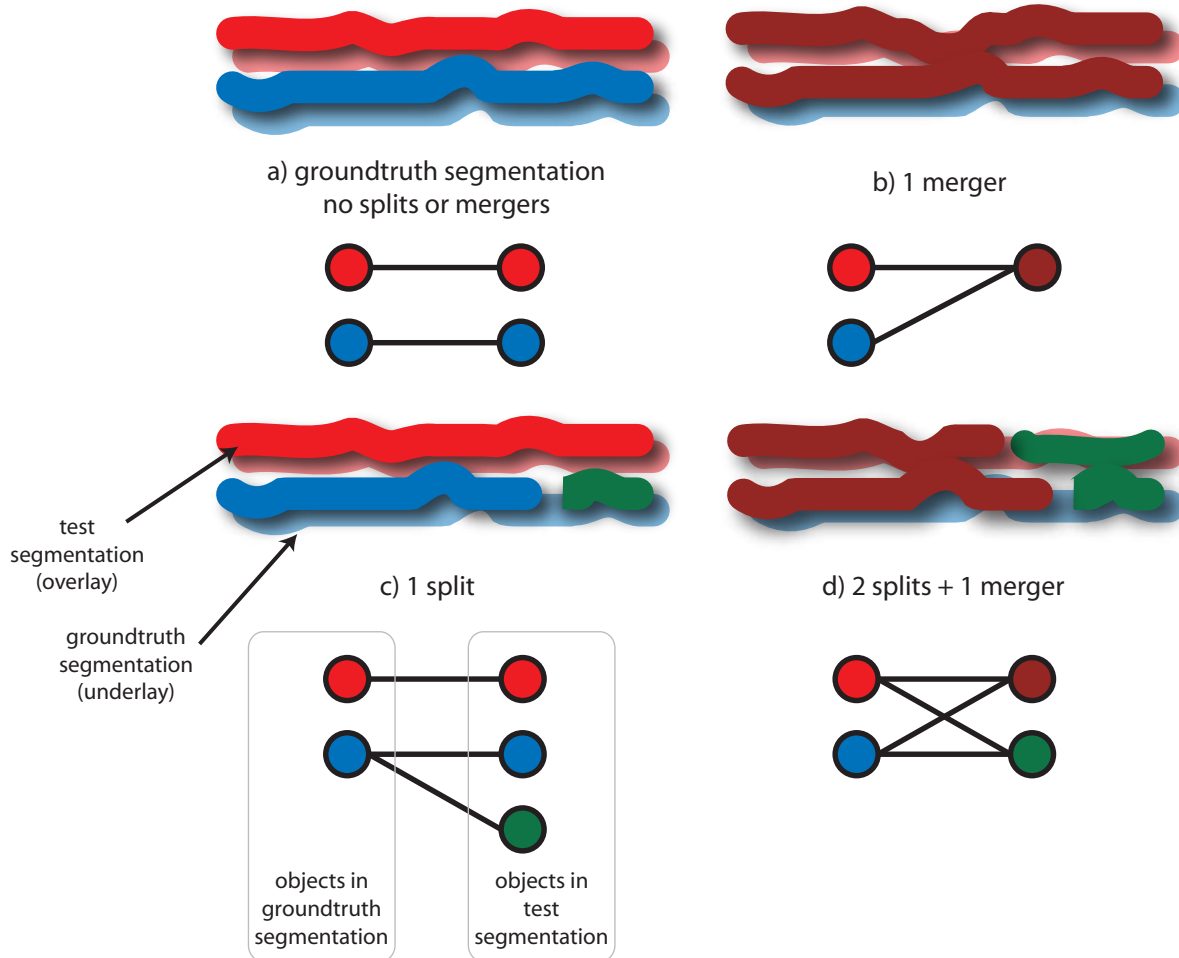
Figure 3-5: Bipartite graph representing splits and mergers. Test segmentation is overlayed on the groundtruth segmentation and the overlap between objects is measured. Overlap codified as a bipartite graph is shown for the example cases of a) perfect segmentation, b) merger, c) split and d) two splits and a merger.

more than one object in the test segmentation. In the overlap graph, this corresponds to a situation where one node in the groundtruth has edges to more than one node in the test segmentation. The number of splits corresponds to the number of edges of a groundtruth node in excess of 1. This may be computed by subtracting the number of groundtruth nodes with edges, from the total number of edges in the bipartite graph.

A merge is said to occur when an object in the groundtruth is erroneously joined with another groundtruth object. This happens when a test object overlaps with more than one groundtruth object. In the overlap graph, this corresponds to nodes in the groundtruth becoming connected via nodes in the test segmentation. A merge has occured between a pair groundtruth objects if there is atleast one node in the test segmentation that has edges to both groundtruth nodes. To compute the number of merges, we first compute distance matrix for all pairs of groundtruth nodes. The number of merges corresponds to the number of pairs of groundtruth 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 Fig. 3-5. 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 groundtruth 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.

3.3.3 Application to measuring skeleton-to-pixel error

Both the Rand index and the split-merge measure can be applied to measuring skeleton-to-pixel segmentation errors in a rather straightforward manner. In both cases, we simply restrict the pixels considered in the error measure to pixels exclusively on the skeleton. When

applied to the Rand index, this yields a measure of the connectivity of pairs of skeleton pixels. When applied to the split-merge metric, this measures the number of skeleton regions split or merged.

Chapter 4

Convolutional networks

Convolutional networks (CN) are a powerful image processing architecture. Classically, they have been developed as a constrained version of regular neural networks for the purpose of handwritten digit recognition. More recently, these networks have been applied successfully for face detection and object recognition. We have developed versions of these networks for image processing.

We use convolutional networks to transform one or more input images into one or more output images. CNs achieve this computation by effectively performing a non-linear filtering of the input image. And since CNs derive from NNs, they inherit the properties of universal function approximation, given sufficient complexity and training data. In the filtering sense, this corresponds to the ability to represent any mapping from an image patch to a pixel.

I will use the nonlinear filtering computation to compute boundary maps and nearest-neighbor affinity graphs corresponding to EM images. In this chapter, I present the mathematical details required for performing inference and for training convolutional networks

4.1 Inference in convolutional networks

In this section, I present the convolutional network formalism. Although CNs are closely related to NNs and are classically derived as constrained NNs, here I present them directly. In a following section, I show how they can be regarded as both a specialization of neural networks and vice versa.
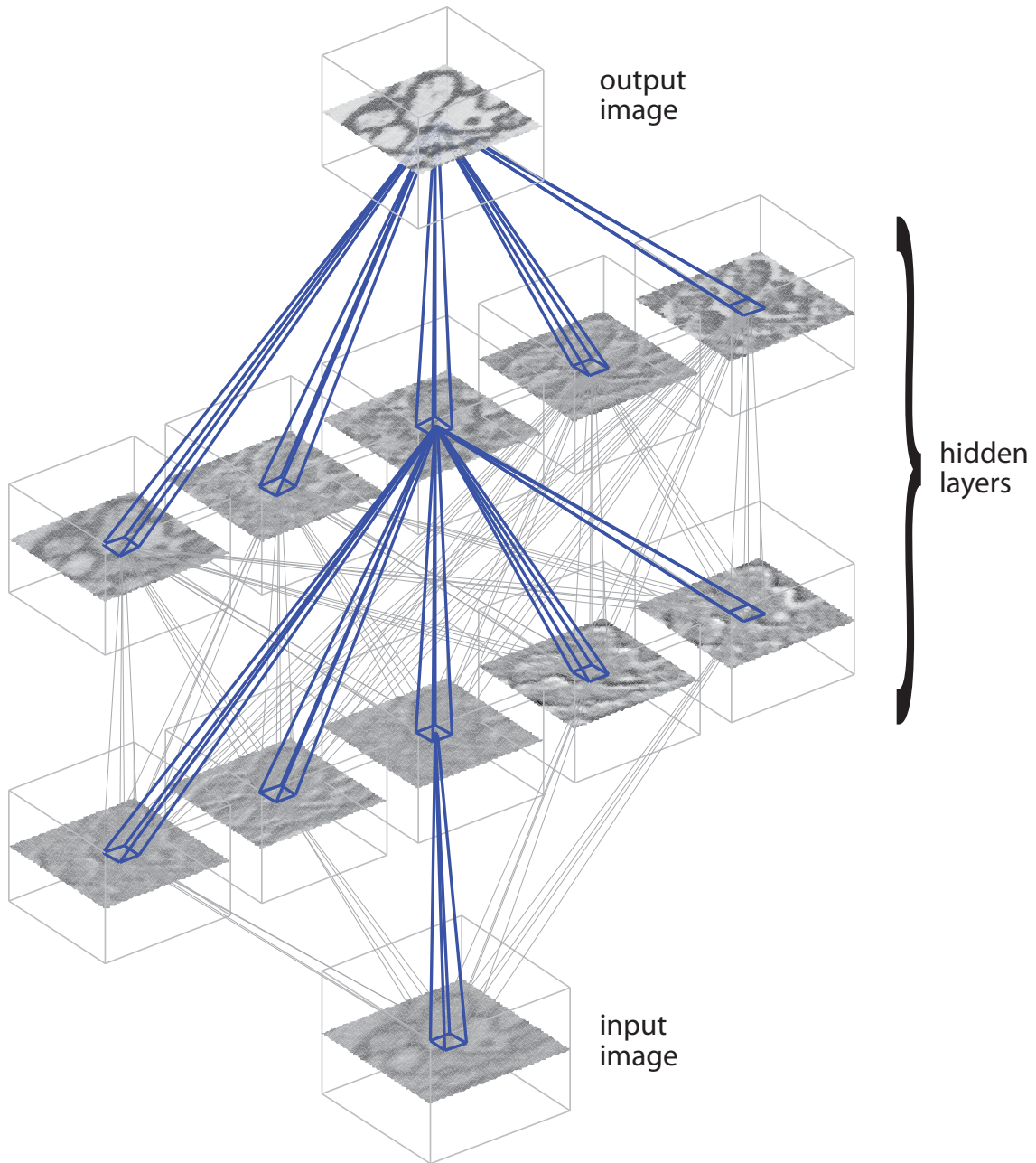
Figure 4-1: The convolutional network transforms a 3d image into one or more 3d images.

3 Hidden Layers
all nodes shown

x edge

Input

y edge

z edge
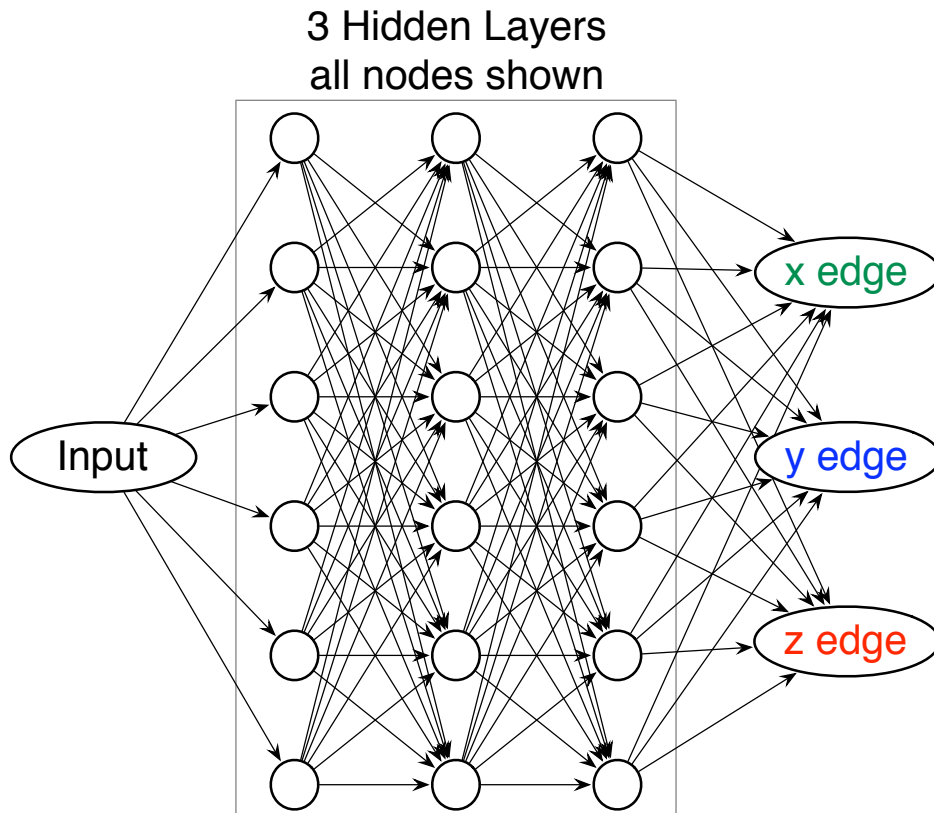
Figure 4-2: The convolutional network architecture can also be represented as a directed graph where each node is an image, and each edge represents convolution by a filter. The CN shown here contains four layers of convolutions with six feature maps each and produces three output images.

Formally, a CN is defined on a directed graph[1] as in Figure 4-2. 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 the equation:

$$I_a = f\left(\sum_b w_{ab} * I_b + h_a\right) \tag{4.1}$$

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 paper 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 (Figure 5-2A). 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.

4.2 Training convolutional networks using gradient descent

The well-known *error backpropagation* algorithm for training NNs can be easily generalized for training w_{ab} and h_a in a CN [LeCun et al., 1989, 1998]. This gradient descent procedure minimizes a cost function that measures the discrepancy between the output image produced

---

[1]The *directed* graph representing a CN must not be confused with the *undirected* affinity graph.

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

$$
\begin{aligned}
S_O &= (I_O - I_O^d) \odot f'(I_O) \\
S_b &= \left(\sum_a S_a \circledast w_{ab} \right) \odot f'(I_b) \\
\Delta w_{ab} &= \eta I_a \circledast S_b \\
\Delta h_a &= \eta \sum_{pixels} S_a
\end{aligned}
\tag{4.2}
$$

These equations implement an efficient recursive gradient computation that gives the back-propagation algorithm its name. While Eq. (4.1) implements a recursive computation where information flows in the direction indicated by the directed edges of the CN graph, the recursion in Eq. 4.2 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}, $\odot$ is a pixel-wise image multiplication operation and $f'(x)$ is the derivative of the nonlinearity $f(x)$. The size of the gradient update at each iteration is controlled by the constant η.

4.3 Some tips and tricks for convolutional network training

To speed up convergence of the gradient procedure, we used some simple tricks from LeCun et al. [1998] which we detail here.

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 (frequently, 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 et al. [1998] that weight vectors have unit norm.

We train the randomly initialized CN using the iterative optimization procedure of stochastic gradient descent with diagonal rescaling [LeCun et al., 1998]. On each iteration, a "mini-batch" was generated by randomly sampling 6x6x6 image patches from the training image. Backpropagation is performed to compute the gradient, which is then rescaled by a diagonal approximation to the Hessian as in LeCun et al. [1998] with a learning rate of 10^{-4}. Training is usually stopped after the training error had plateau-ed. There seems to rarely be a need for cross-validation as little overfitting was ever observed in our networks.

4.4 Comparison to previous work on convolutional networks

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, etc.) 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 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 which may have no boundary pixels separating them (in contrast to [Ning et al., 2005], which would require atleast 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.

4.5 Comparing convolutional and neural networks

4.5.1 Neural networks and image processing

Conventionally, neural networks 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 a 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.

There are several ways in which a neural network may be applied to an image processing problem. In the simplest case, the pixels of an entire image may be re-ordered 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-valued, but the output is scalar-valued or vector-valued. A second approach is more suitable to non-linear 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 a 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 Fig 4-3.

4.5.2 Patch-based neural networks are a special case of convolutional networks

Using the terminology developed in 4, we can represent any patch-based NN classifier as 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 a 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 corresponds exactly to scalars given by $W^{\alpha\beta}$.

Thus the crucial difference between a NN and a CN is in the nature of the w_{ab} parameters. A 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 more powerful generalization of a NN in a manner appropriate for image processing applications since convolutions are allowed at all stages in the network.

4.5.3   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}.

$$I_a = f\left(\sum_b W^{ab} I_b + h_a\right) \tag{4.3}$$
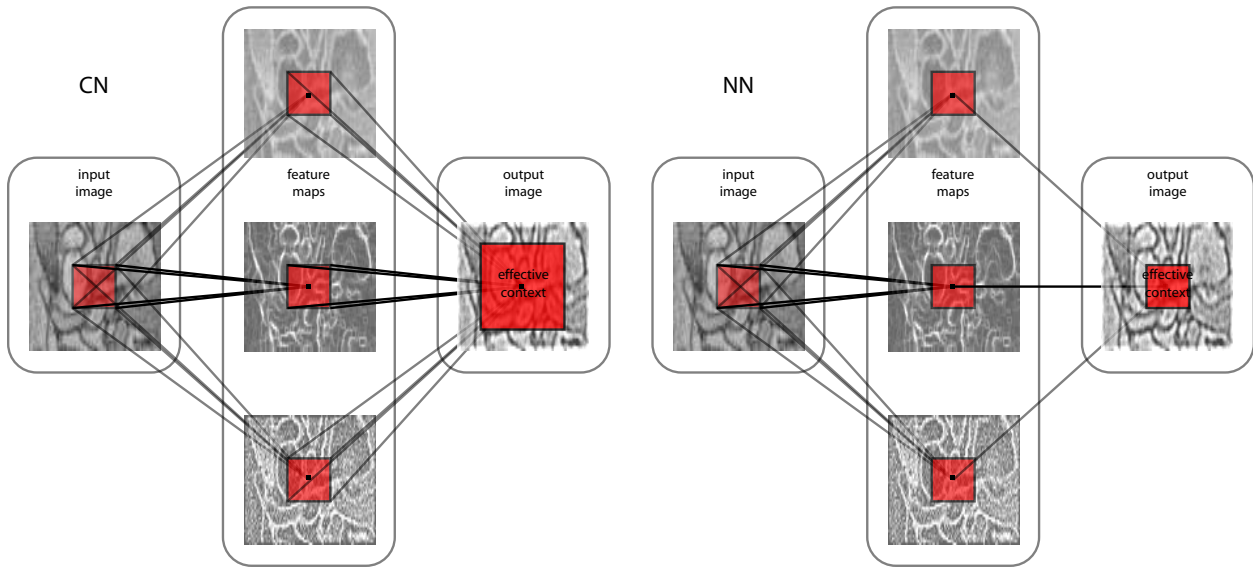
Figure 4-3: A convolutional network constructs progressively larger as well as more complex image features. A neural network can only construct more complex features at each layer, but not larger features.

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 becomes complicated and less useful for a CN with more than one hidden unit in each layer.

Chapter 5

Learning nearest neighbor affinity graphs with convolutional networks

Abstract

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

5.1   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 and Malik, 2000, Fowlkes et al., 2003]. In this paper, 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 datasets. 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 network*s. 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 and Horstmann, 2004). This technique produces 3d images with a spatial resolution of roughly 30 nm or better (Figure 5-1A). If applied to a tissue sample of just 0.5 mm x 0.5 mm x 1.2 mm from the cerebral cortex, the method would yield a 3d image of several

56

tens of teravoxels. Such a volume would contain about 15,000 neurons and many thousands of non-neuronal 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 and Denk, 2006]. The sheer volume and complexity of data to be segmented precludes manual reconstruction and makes 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 et al., 1986, Fiala, 2005] to semi-automated interactive methods [Carlbom et al., 1994] and fully automated methods [Mishchenko, 2009, Jurrus et al., 2006, Helmstaedter et al., 2008, Andres et al., 2008]. White et al. [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 [Al-Kofahi et al., 2002, Jurrus et al., 2006, Mishchenko, 2009, Andres et al., 2008]. For example, Jurrus et al. [Jurrus et al., 2006] 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. [Carlbom et al., 1994] adapted the widely-used *active contour* (snakes) technique for semi-automated 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. [Helmstaedter et al., 2008] use supervised learning in the form of a nearest neighbor classifier to segment 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].

5.1.1 Organization
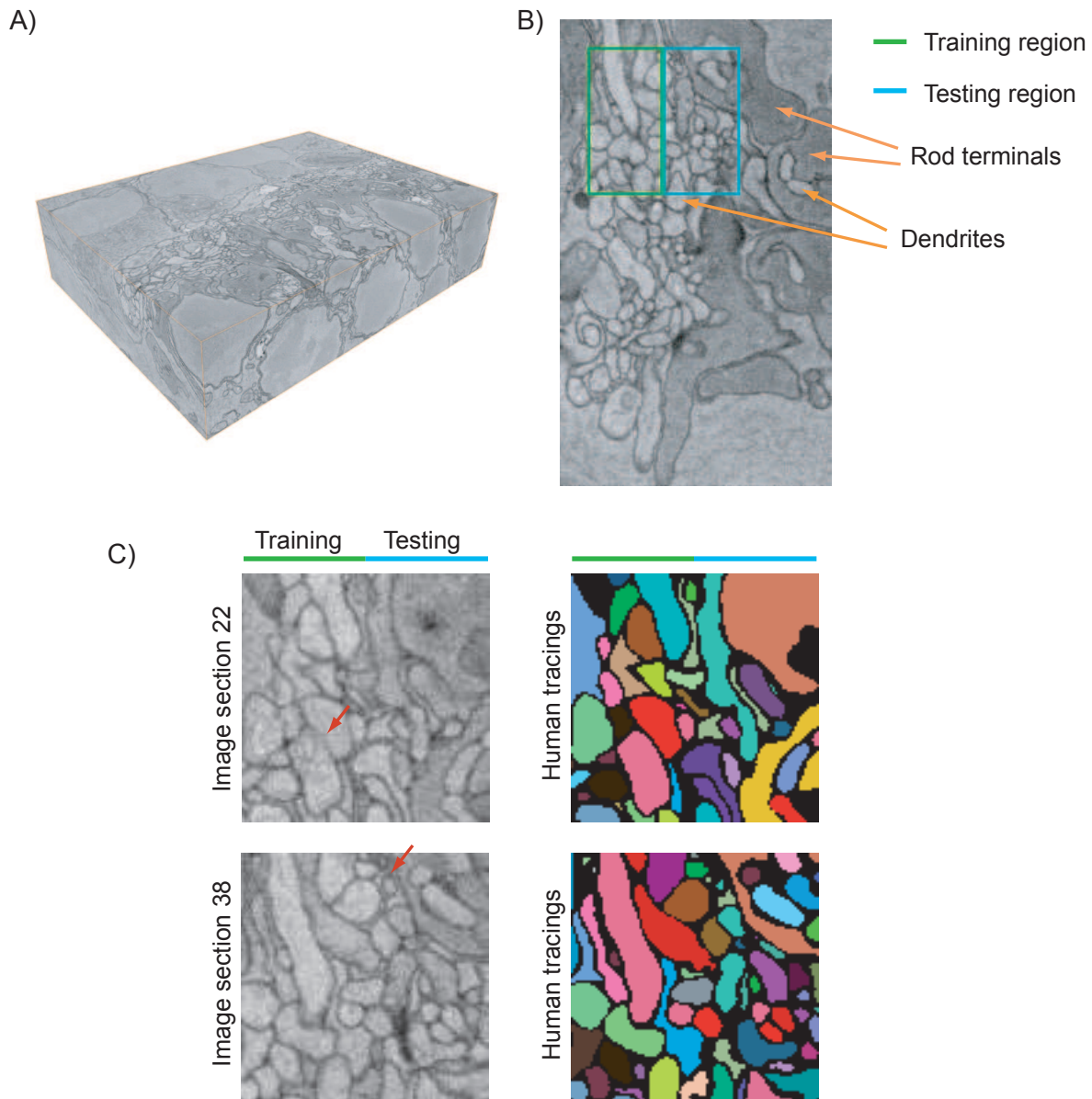
Figure 5-1: A) View of 3d stack of images generated by serial block-face scanning electron microscopy (SBF-SEM). Tissue is from the outer plexiform layer of the rabbit retina. The total volume is 800x600x100 voxels corresponding to 20.96x15.72x5 μm^3. B) Larger view of a section from this volume. The region traced by humans is divided into training and test sets, shown in green and blue. Some of the large gray objects are axons 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 100x100x100 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.

The paper is organized as follows. We first define convolutional networks in Section 4. Section 5.2 describes the structure of the affinity graph, and how we apply convolutional networks to produce this graph directly from the raw image. In Section 5.3 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 5.4 with some future directions, and the two appendices give algorithmic details and performance metrics.

5.2 Using convolutional networks to generate an affinity graph

Figure 5-2 illustrates the computational problem solved in this paper 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 5-2A. Since each edge is shared by a pair of nodes, there are three times as many edges as nodes. As shown in Figure 5-2A, we can think of the affinity graph as 3 different images, each representing the affinities of the edges of a particular direction. Therefore, the problem can be seen as 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 which correspond to different image segments. The computational problem of segmentation thus involves two steps: first the transformation of the input image into an affinity graph, and second 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 be-

tween nodes in the same segment have value 1 and edges between nodes not contained in the same segment have value 0 (see Figure 5-2C). 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.

5.3   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 more simple 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 inter-human 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 runs much faster than normalized cuts.

5.3.1   About the dataset

We use images of retinal tissue collected with the serial block-face scanning electron microscope (SBF-SEM; Denk and Horstmann, 2004). We imaged a volume of dimension 21x15.6x5 μm^3, corresponding to 800x600x100 voxels at a resolution of 26x26x50 nm^3 (Figure 5-1A). Cell bodies compose the majority of the volume at the sides, while in the center, much smaller neurites are more common. To focus on neurites, a central region of 320x200x100 voxels was selected for investigating automated segmentations (Figure 5-1B). A subvolume

of 100x100x100 voxels was completely segmented by two human experts who traced the contours of all objects within this volume (Figure 5-1C). These segmentations were converted into desired affinity graphs, as described in Section 5.2, 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 between experts on the exact placements of boundaries. This variability is used in our analysis to suggest a baseline inter-human error rate. Figure 5-1C highlights some of the difficult regions typical of those encountered in this data (red arrows). In z section 22 (upper 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 little as 4 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.

5.3.2 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 Chapter 4.

Our CN has an architecture with three hidden layers, each containing 6 feature maps. All filters in the CN are of size 5x5x5, but since CN contains four convolutions between input and output, a single output voxel is a function of a 17x17x17 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 5-3A, showing that approximately 90% of the edges are classified correctly. Figure 5-3B quantifies the performance at edge classification

using a precision-recall curve [Martin et al., 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 Chapter 3. Here it's 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 one 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 5-3B, 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 4,913 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 megavoxels. 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 only used learning to optimize the combinations of these features. For example, Fowlkes et al. [Fowlkes et al., 2003] use classifiers with 7 inputs and only a few dozens of free parameters. Our approach shows that good performance can be achieved without the careful design and selection of image features.

5.3.3   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.

$$NCut(\{\mathcal{V}_1, \ldots, \mathcal{V}_k\}) = \min_{\mathcal{V}_1, \ldots, \mathcal{V}_k} \sum_{i=1}^{k} \frac{\text{links}(\mathcal{V}_i, \mathcal{G} \backslash \mathcal{V}_i)}{\text{degree}(\mathcal{V}_i)} \tag{5.1}$$

where $\mathcal{G} \backslash \mathcal{V}_i$ is the set of vertices not in $\mathcal{V}_i$. The number $\text{links}(\mathcal{V}_i, \mathcal{G} \backslash \mathcal{V}_i)$ is usually referred to as the *cut value* since it 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 5.1 is known to be NP-hard, Shi and Malik suggested a polynomial time approximation based on spectral methods [Shi and Malik, 2000]. 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 multi-scale kernel weighted k-means algorithm, which finds local minima of the normalized cuts cost function [Dhillon et al., 2005]. For large problems such as ours ($>$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 running the algorithm, 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.

5.3.4 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, but only an intermediate step towards a segmentation. Therefore we evaluated the CN affinity graphs by applying the popular normalized cuts algorithm to partition the affinity graph

into segments (Appendix 5.3.3). It bears repeating, however, that our affinity graph can be paired with any graph partitioning algorithm.

Figure 5-4 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 5-5. In comparing the segmentation errors of the normalized cuts algorithm using the standard affinity with the CN affinity (NCUT-STD vs. 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 vs. human segmentations with the same metrics (Human-Human).

Note that normalized cuts requires 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.

5.3.5 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 Appendix 5.3.3, 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 which does not require the solution of a complex 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 et al., 1990]. 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 under-segmentation of the image. The performance of this segmentation algorithm using the CN affinity can be seen for various thresholds in Figure 5-5. 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 5-6, comparing the original human segmentation to CN+CC and CN+NCUT. In Figure 5-6A, the large branching component is segmented correctly by both algorithms, while in B, both algorithms split the component at narrow regions (CC preserves slightly more of the object), in C, the thin branching component is correctly segmented.

5.4 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 post-processing 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 post- processing.

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 and 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/no-vessel decision at each pixel, and post-processing 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-/post- processing makes the computation more efficient and relies on less training data. We essentially take an opposite view: that 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/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 target metric, and so are less likely to throw away valuable information than hand-designed pre-processing.

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 only has one stage of spatial filtering, 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 the Appendix. 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 Fig. 4-3, 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.

5.4.1   Efficient graph partitioning

There has been much research in recent years on developing good graph partitioning criteria and algorithms [Shi and Malik, 2000, Felzenszwalb and Huttenlocher, 2004, Boykov et al., 2001, Cour et al., 2005a, Gdalyahu et al., 1999]. Continuing our theme of relying more on learning methods and less on elaborate post-processing, 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) which 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 works 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 towards 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 pre-existing 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., 2005a]. The extension of our learned affinity functions to more edges is left for future work.

5.4.2 Progress towards neural circuit reconstruction using serial-section EM

We have made a first step towards automated reconstructing of neural circuits using images from serial block-face scanning electron microscopy (SBF-SEM; Denk and Horstmann, 2004). Figure 5-7 shows the 3d reconstruction of selected neural processes in the image volume, confirming that the generated segments are biologically realistic. Figure 5-8 elaborates on this by showing the largest 100 components in the 320x200x100 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 which 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 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.

5.5 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 (Figure 4-2) 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 7x7x7. The filters connecting the feature maps in the hidden layer to the output images are 1x1x1 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 non-linearity. In contrast to our CN, this network requires more than twice as many free parameters (25,876) to process an input patch of only 7x7x7. Our attempts to train NN-like networks with larger filters was plagued with poor generalization performance and with gradient-learning optimizing to poor local minima in parameter space.

From Figure 5-10, 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 (17x17x17 versus 7x7x7), 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 importantly, CN makes fewer split and merge errors, leading to superior segmentation performance.

While the CN and NN can both 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.
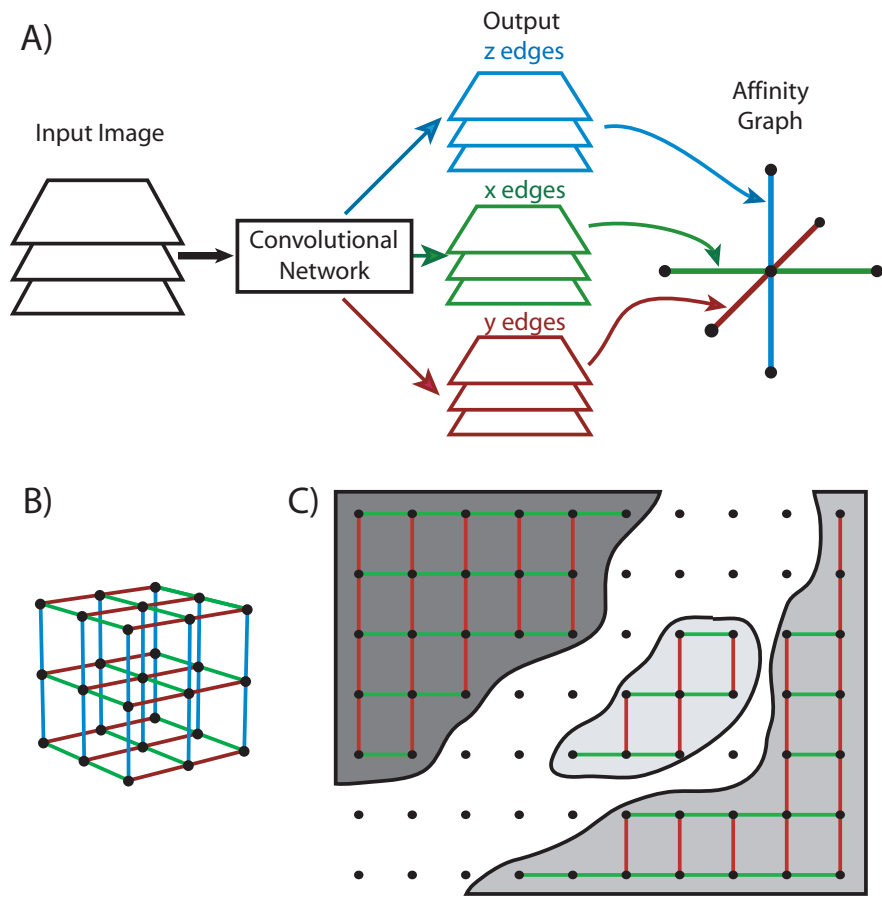
Figure 5-2: A) Creating the affinity graph using a convolutional network. The input to the network is the 3d EM image and the desired output is a set of 3d images: one each for the x, y and z directions representing the affinity graph. B) The edges of the nearest-neighbor affinity graph form a lattice. 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".
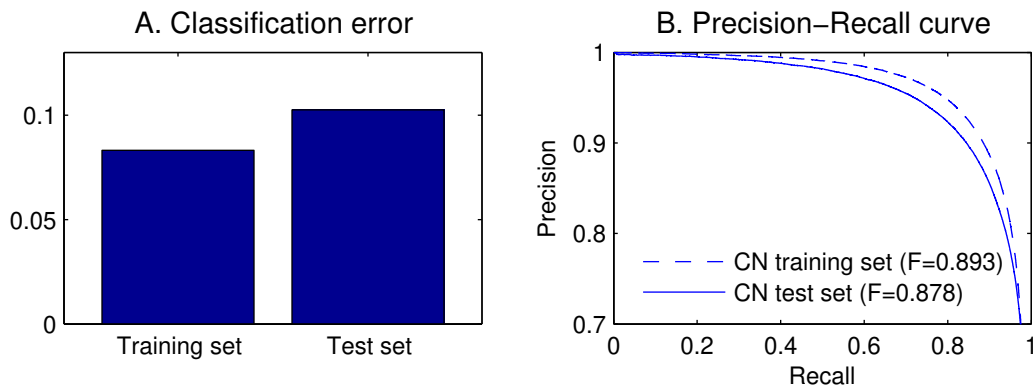
Figure 5-3: Performance of the convolutional network at predicting affinity between individual voxels. A) Approximately 10% of the affinities are misclassified in the test set. B) A more thorough quantification of the continuous-valued CN output using precision-recall curves (see Appendix) as in Martin et al. [2004] shows good performance (higher F-scores and curves closest to the upper right are superior).
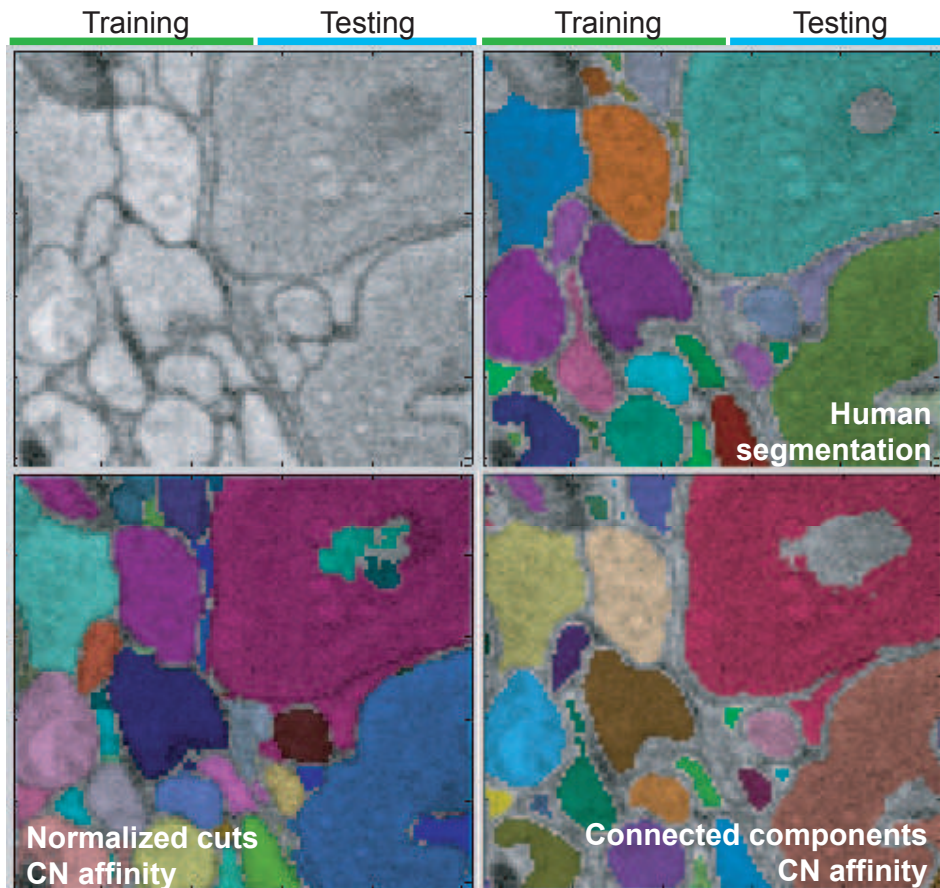
Figure 5-4: Top row: Original EM images and human-labeled segmentation, where different colors correspond to different segments. Bottom left: Using normalized cuts on the output of the CN network qualitatively segments many of the objects correctly. Bottom right: The connected components procedure using CN affinity creates segmentations that are structurally superior to normalized cuts with fewer splits and mergers, and the segments are shrunken within the intracellular regions.
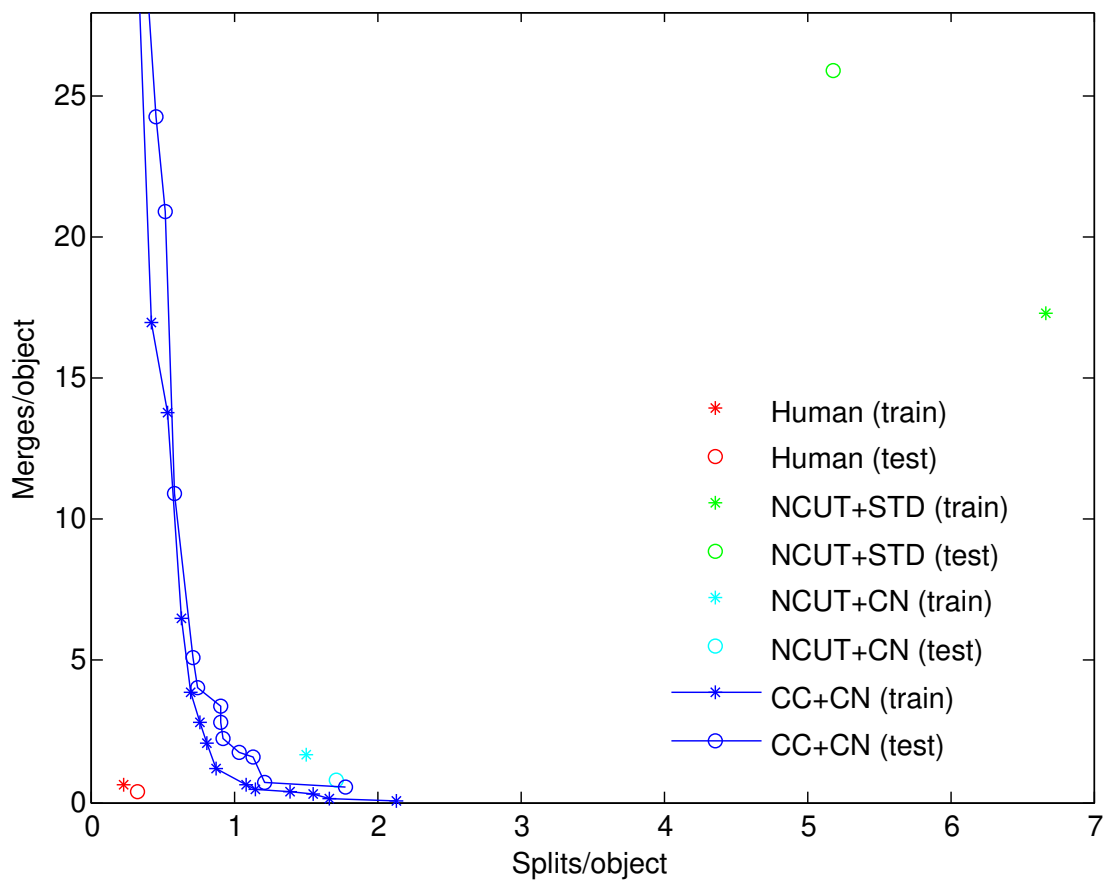
Figure 5-5: Segmentation performance is quantified by measuring the number of splits and merges per true segment. Points closer to the origin have lower segmentation error. Circles and asterisks are used to denote the test and training set errors, respectively. CC+STD has split-merge errors of (121.9,15.7) 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 over-segmentation.
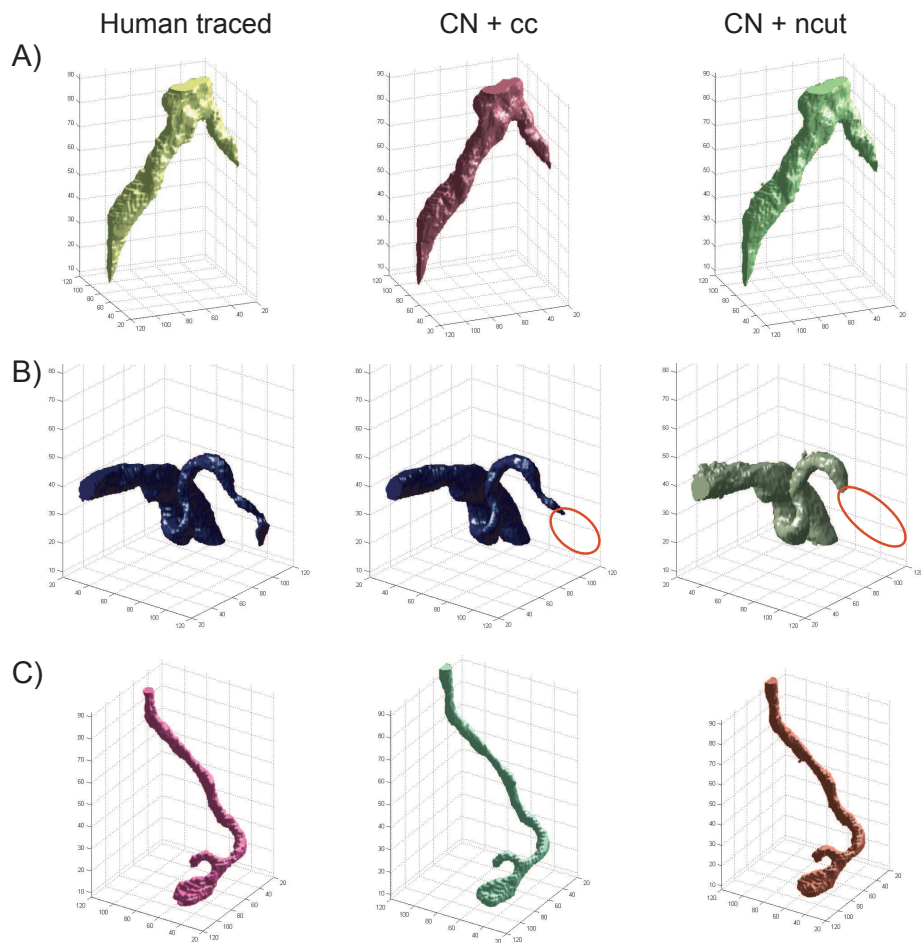
Figure 5-6: Components traced by human (left) compared with automated results, convolutional network with connected component segmentation CN-CC (middle), and with normalized cut segmentation CN-NCUT (right). A) Correctly reconstructed object, showing close agreement between human tracings and algorithm output. B) An object which has thin processes which are 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.
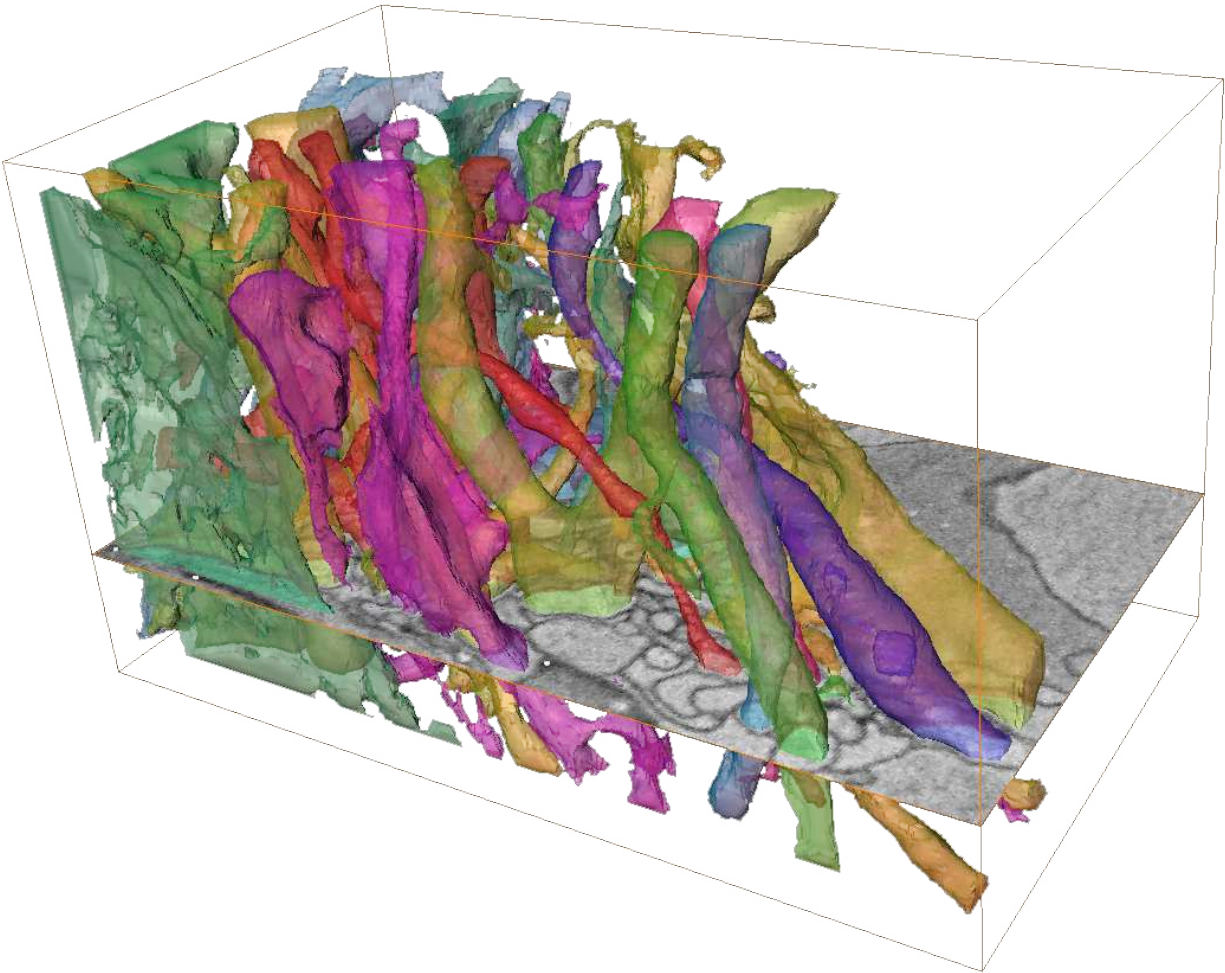
Figure 5-7: 3d reconstructions of selected processes using our algorithm (CN+CC). Large cell bodies are not shown to avoid hiding smaller processes.

Components 2-10   11-20   21-40

41-60   61-80   81-100
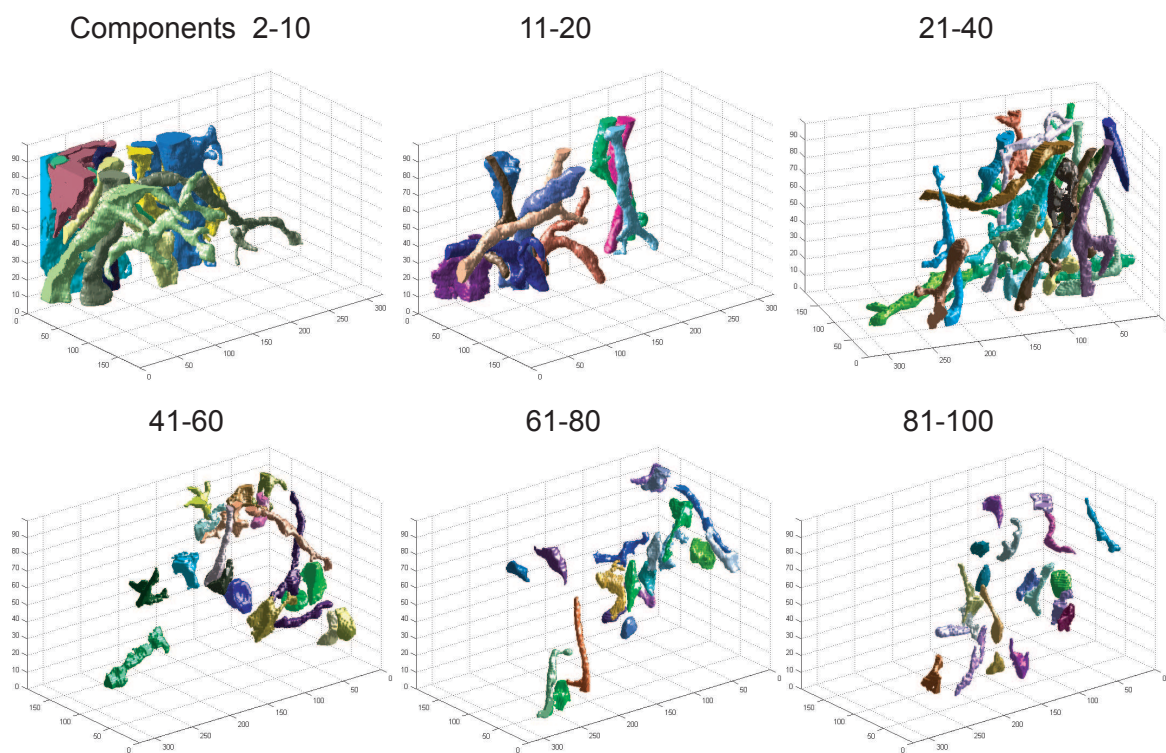
Figure 5-8: The 100 largest components in a 320x200x100 volume (7.86 μm x 5.24 μm x 5μm), omitting cell bodies and large glial processes for clarity.
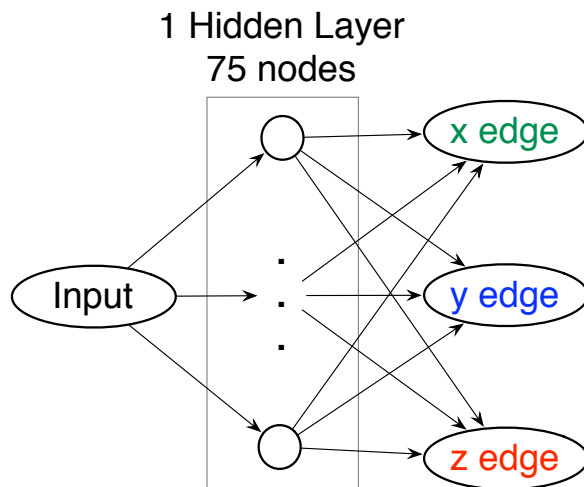
Figure 5-9: A restricted convolutional network that is equivalent to a two layer neural network. Only the first layer of the network is convolutional; edges from the input image to the hidden nodes represent convolution filters, while edges from the hidden units to the output units are scalar valued. This network has many more nodes in the hidden layer than our CN, but has fewer layers.
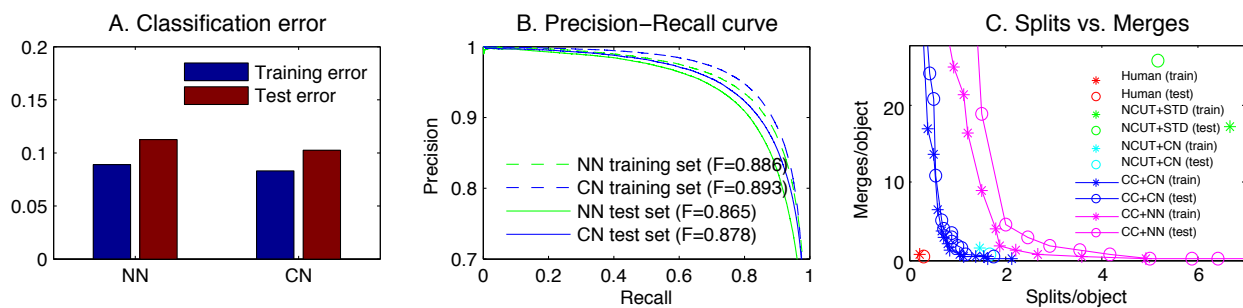
Figure 5-10: CN and NN have similar performance at affinity prediction, but CN has superior segmentation performance. A) Approximately 10% of the affinities are misclassified in the test set. B) A more thorough quantification of the continuous-valued affinity output using precision-recall curves as in Fowlkes et al.[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).

Chapter 6

Ultrametricity and hierarchy learning

In this chapter I introduce the concept of an *ultrametric* distance function, its relationship with hierarchical clustering and a means for learning it. Image segmentation can be represented as a clustering problem, so this formalism will be useful for constructing an algorithm for the direct end-to-end machine learning of image segmentation.

6.1 Metric and ultrametric distance functions

A *metric* on a set X is a function $d : X \times X \rightarrow \mathcal{R}$ satisying the following properties

$$
\begin{align}
d(x, y) &\geq 0 \quad \text{with equality if and only if } x = y \tag{6.1} \\
d(x, y) &= d(y, x) \tag{6.2} \\
d(x, y) &\leq d(x, z) + d(y, z) \tag{6.3}
\end{align}
$$

Here, the first two properties require positive definiteness and symmetry of the distance function and the third is the triangle inequality.

An *ultrametric* distance function is a metric distance that satisfies a stronger version of the triangle inequality.

$$
d(x, y) \leq \max(d(x, z), d(y, z)) \tag{6.4}
$$

Therefore all ultrametrics are metric, but not vice versa.

Ultrametric spaces have some non-intuitive properties deriving from the conditions above.

Proposition 6.1. *All triangles are isosceles. But not all isosceles triangles are possible.*

Recall that triangles with two sides of equal length are called isosceles. The repeated application of the ultrametric triangle inequality to the same three points with x, y and z relabeled implies that all triangles formed by any three points in an ultrametric space are isosceles. Further, the length of the two equal sides of the triangle must be greater than or equal to the length of the third side.

Proposition 6.2. *Every point inside a ball of finite radius is at its center.*

Given a ball centered around x of radius r defined by $B(x; r) = \{y | d(x, y) < r\}$, for every point y contained in $B(x; r)$, $B(x; r) = B(y; r)$. This surprising fact is related to the strong triangle inequality due to which the diameter of a ball is the same as its radius.

Proposition 6.3. *Whenever two balls intersect, one ball is entirely contained within the other.*

By the strong triangle inequality, whenever two balls $A(x, r)$ and $B(y, r')$ intersect, where the radius of ball r' is larger than r, there can never be a partial intersection where only some of the points in ball A are contained within ball B. We can prove this by contradiction. Assume point z is the only point contained in $A \cap B$. This would require that $d(x, y) \not\leq \max(d(x, z), d(y, z))$ which is impossible in an ultrametric space.

Taken together, Propositions 6.2 and 6.3 imply a rather peculiar but extremely useful topology. Since balls of the same radius either never intersect or overlap completely, they form a unique partitioning or clustering of points in an ultrametric space.

These facts together imply that an ultrametric distance defined over a set of points can be represented uniquely as a dendrogram.

6.2   Ultrametricity, hierarchies and dendrograms

There exists a bijection between the ultrametric distances on a finite set of points and a dendrogram-based hierarchical clustering of the set. Thus ultrametric spaces can be used to exactly represent hierarchical partitionings.

**nesting property of balls
in ultrametric spaces**

partial intersection   *no intersection*   *complete intersection*

✗   ✔   ✔

*forbidden by
strong triangle inequality*
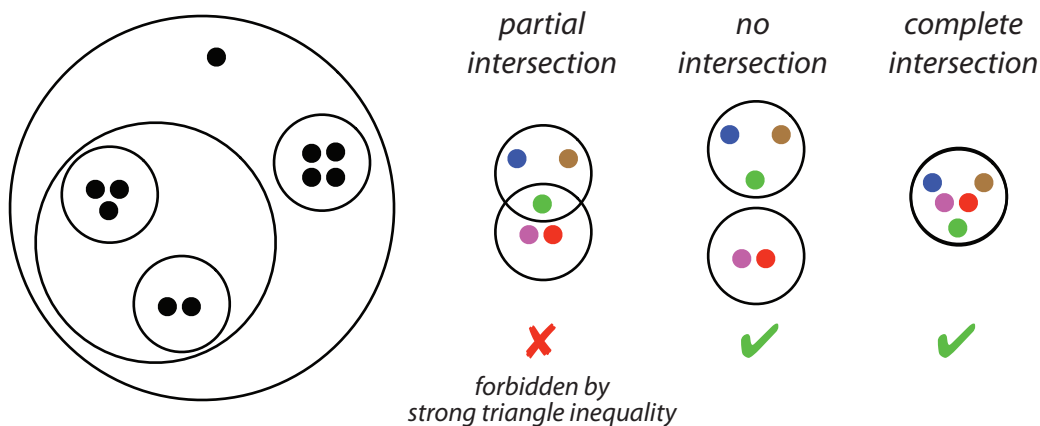
Figure 6-1: Balls in an ultrametric space form a nested partitioning of points due to the strong triangle inequality. The strong triangle inequality prohibits partially overlapping balls in an ultrametric space.

6.2.1   Constructing dendrograms from ultrametric distances

Given a set of points and ultrametric distances between them, one can construct a unique dendrogram by use of the nesting property of balls in an ultrametric space (Fig. 6-1).

6.2.2   Ultrametric distance corresponding to a dendrogram

Dendrograms can be used to define ultrametric distances. Recall from 2 that a dendrogram is representation of a hierarchical clustering. It is a rooted tree graph whose leaf nodes correspond to points being clustered and whose internal nodes represent groups of objects.

Definition 6.4. The tree distance between two points is the height of their earliest common ancestor node.

Proposition 6.5. *Tree distances on a dendrogram are ultrametric.*

It can be easily verified that the ultrametricity conditions hold for the tree distance.

6.3   Hierarchy learning by learning ultrametric distances
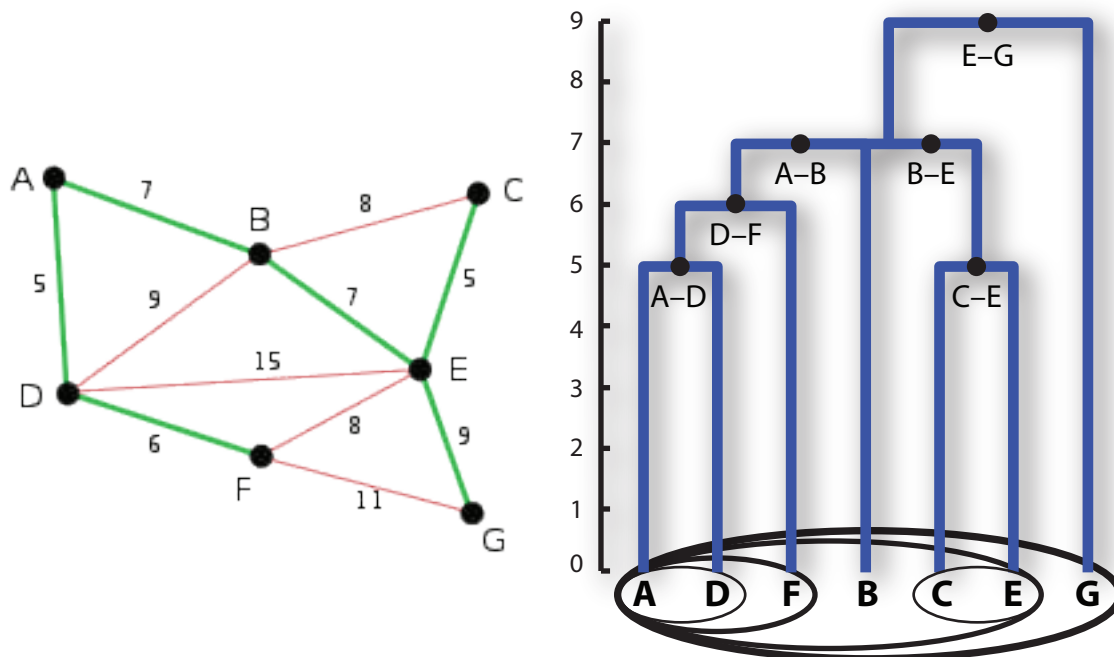
Figure 6-2: There is a one-to-one correspondence between ultrametric distances on a set of points and a dendrogram.

Metric learning has long been used as a technique for the supervised learning of clustering. Given a parametrized distance function, the idea is to adjust the parameters such that points belonging to the same cluster are close together and points belonging to different clusters are far apart. However, general distance functions are not consistent in the following sense. Even though two points may be far apart from each other, there may be a high density path of points between which makes the interpretation of the clustering dependent not only on the distances between the two points but also on the distances between all the other points in the dataset.

Ultrametric distances have the useful property of "transitive consistency". If points a and b are far apart, and c is close to a, then c is also far from b. Conversely, if a and b are close to each other, and c is close to a, then a, b and c are all close to each other. This property does not hold for general metric distances and leads to inconsistent partitioning cues from the distance function.

We suggest that learning of ultrametric distances is more appropriate for clustering problems than general metric distances. In the next chapter, we train minimax graph distances

in order to learn the clustering problem of image segmentation.

Chapter 7

Learning image segmentation using ultrametric learning

Abstract

Images can be segmented by first using a classifier to predict an affinity graph that reflects the degree to which image pixels must be grouped together and then partitioning the graph to yield a segmentation. Machine learning has been applied to the affinity classifier to produce affinity graphs that are good in the sense of minimizing edge misclassification rates. However, this error measure is only indirectly related to the quality of segmentations produced by ultimately partitioning the affinity graph. We present the first machine learning algorithm for training a classifier to produce affinity graphs that are good in the sense of producing segmentations that directly minimize the Rand index, a well known segmentation performance measure.

   The Rand index measures segmentation performance by quantifying the classification of the connectivity of image pixel pairs after segmentation. By using the simple graph partitioning algorithm of finding the connected components of the thresholded affinity graph, we are able to train an affinity classifier to directly minimize the Rand index of segmentations resulting from the graph partitioning. Our learning algorithm corresponds to the learning of maximin affinities between image pixel pairs, which are predictive of the pixel-pair connectivity.

7.1   Introduction

Supervised learning has emerged as a serious contender in the field of image segmentation,

ever since the creation of training sets of images with "ground truth" segmentations provided

by humans, such as the Berkeley Segmentation Dataset Martin et al. [2001]. Supervised learning requires 1) a parametrized algorithm that map images to segmentations, 2) an objective function that quantifies the performance of a segmentation algorithm relative to ground truth, and 3) a means of searching the parameter space of the segmentation algorithm for an optimum of the objective function.

In the supervised learning method presented here, the segmentation algorithm consists of a parametrized *classifier* that predicts the weights of a nearest neighbor affinity graph over image pixels, followed by a graph *partitioner* that thresholds the affinity graph and finds its connected components. Our objective function is the Rand index Rand [1971], which has recently been proposed as a quantitative measure of segmentation performance Unnikrishnan et al. [2007]. We "soften" the thresholding of the classifier output and adjust the parameters of the classifier by gradient learning based on the Rand index.

Because maximin edges of the affinity graph play a key role in our learning method, we call it *maximin affinity learning of image segmentation*, or MALIS. The minimax path and edge are standard concepts in graph theory, and maximin is the opposite-sign sibling of minimax. Hence our work can be viewed as a machine learning application of these graph theoretic concepts. MALIS focuses on improving classifier output at maximin edges, because classifying these edges incorrectly leads to genuine segmentation errors, the splitting or merging of segments.

To the best of our knowledge, MALIS is the first supervised learning method that is based on optimizing a genuine measure of segmentation performance. The idea of training a classifier to predict the weights of an affinity graph is not novel. Affinity classifiers were previously trained to minimize the number of misclassified affinity edges Fowlkes et al. [2003], Martin et al. [2004]. This is not the same as optimizing segmentations produced by partitioning the affinity graph. There have been attempts to train affinity classifiers to produce good segmentations when partitioned by normalized cuts Meila and Shi [2001], Bach and Jordan [2006]. But these approaches do not optimize a genuine measure of segmentation performance such as the Rand index. The work of Bach and Jordan Bach and Jordan [2006] is the closest to our work. However, they only minimize an upper bound to a renormalized version of the Rand index. Both approaches require many approximations to make the learning tractable.

84

In other related work, classifiers have been trained to optimize performance at detecting image pixels that belong to object boundaries Martin et al. [2004], Dollár et al. [2006], Maire et al. [2008]. Our classifier can also be viewed as a boundary detector, since a nearest neighbor affinity graph is essentially the same as a boundary map, up to a sign inversion. However, we combine our classifier with a graph partitioner to produce segmentations. The classifier parameters are not trained to optimize performance at boundary detection, but to optimize performance at segmentation as measured by the Rand index.

There are also methods for supervised learning of image labeling using Markov or conditional random fields He et al. [2004]. But image labeling is more similar to multi-class pixel classification rather than image segmentation, as the latter task may require distinguishing between multiple objects in a single image that all have the same label.

In the cases where probabilistic random field models have been used for image parsing and segmentation, the models have either been simplistic for tractability reasons Kumar and Hebert [2003] or have been trained piecemeal. For instance, Tu et al. Tu et al. [2005] separately train low-level discriminative modules based on a boosting classifier, and train high-level modules of their algorithm to model the joint distribution of the image and the labeling. These models have never been trained to minimize the Rand index.

7.2   Partitioning a thresholded affinity graph by connected components

Our class of segmentation algorithms is constructed by combining a classifier and a graph partitioner (see Figure 7-1). The classifier is used to generate the weights of an affinity graph. The nodes of the graph are image pixels, and the edges are between nearest neighbor pairs of pixels. The weights of the edges are called affinities. A high affinity means that the two pixels tend to belong to the same segment. The classifier computes the affinity of each edge based on an image patch surrounding the edge.

The graph partitioner first thresholds the affinity graph by removing all edges with weights less than some threshold value θ. The connected components of this thresholded
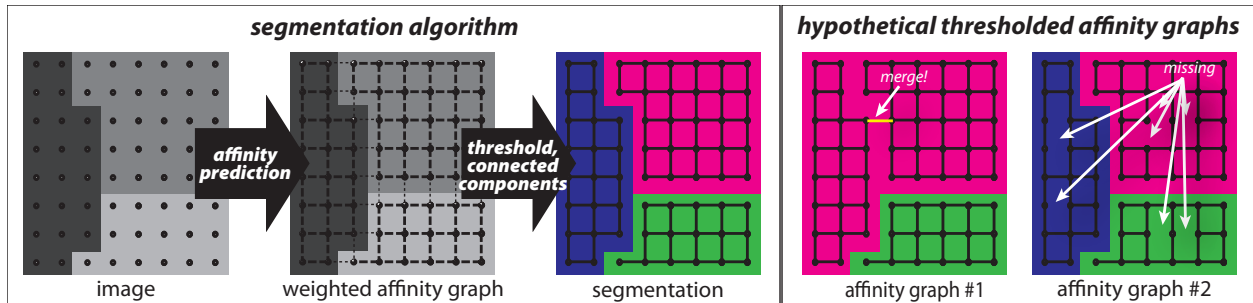
Figure 7-1: (left) **Our segmentation algorithm.** We first generate a nearest neighbor weighted affinity graph representing the degree to which nearest neighbor pixels should be grouped together. The segmentation is generated by finding the connected components of the thresholded affinity graph. (right) **Affinity misclassification rates are a poor measure of segmentation performance.** Affinity graph #1 makes only 1 error (dashed edge) but results in poor segmentations, while graph #2 generates a perfect segmentation despite making many affinity misclassifications (dashed edges).

affinity graph are the segments of the image.

For this class of segmentation algorithms, it's obvious that a single misclassified edge of the affinity graph can dramatically alter the resulting segmentation by splitting or merging two segments (see Fig. 7-1). This is why it is important to learn by optimizing a measure of segmentation performance rather than affinity prediction.

We are well aware that connected components is an exceedingly simple method of graph partitioning. More sophisticated algorithms, such as spectral clustering Shi and Malik [2000] or graph cuts Boykov et al. [2001], might be more robust to misclassifications of one or a few edges of the affinity graph. Why not use them instead? We have two replies to this question.

First, because of the simplicity of our graph partitioning, we can derive a simple and direct method of supervised learning that optimizes a true measure of image segmentation performance. So far learning based on more sophisticated graph partitioning methods has fallen short of this goal Meila and Shi [2001], Bach and Jordan [2006].

Second, even if it were possible to properly learn the affinities used by more sophisticated graph partitioning methods, we would still prefer our simple connected components. The classifier in our segmentation algorithm can also carry out sophisticated computations, if its representational power is sufficiently great. Putting the sophistication in the classifier has the advantage of making it learnable, rather than hand-designed.

86

The sophisticated partitioning methods clean up the affinity graph by using prior assumptions about the properties of image segmentations. But these prior assumptions *could be incorrect.* The spirit of the machine learning approach is to use a large amount of training data and minimize the use of prior assumptions. If the sophisticated partitioning methods are indeed the best way of achieving good segmentation performance, we suspect that our classifier will learn them from the training data. If they are not the best way, we hope that our classifier will do even better.

7.3 Connectivity and maximin affinity

Recall that our segmentation algorithm works by finding connected components of the thresholded affinity graph. Let $\hat{S}$ be the segmentation produced in this way. To apply the Rand index to train our classifier, we need a simple way of relating the indicator function $\delta(\hat{s}_i, \hat{s}_j)$ in the Rand index to classifier output. In other words, we would like a way of characterizing whether two pixels are connected in the thresholded affinity graph.

To do this, we introduce the concept of maximin affinity, which is defined for any pair of pixels in an affinity graph (the definition is generally applicable to any weighted graph). Let A_{kl} be the affinity of pixels k and l. Let $\mathcal{P}_{ij}$ be the set of all paths in the graph that connect pixels i and j. For every path P in $\mathcal{P}_{ij}$, there is an edge (or edges) with minimal affinity. This is written as $\min_{\langle k,l \rangle \in P} A_{kl}$, where $\langle k,l \rangle \in P$ means that the edge between pixels k and l are in the path P.

A maximin path P_{ij}^* is a path between pixels i and j that maximizes the minimal affinity,

$$P_{ij}^* \;=\; \arg\max_{P \in \mathcal{P}_{ij}} \min_{\langle k,l \rangle \in P} A_{kl} \tag{7.1}$$

The maximin affinity of pixels i and j is the affinity of the maximin edge, or the minimal affinity of the maximin path,

$$A_{ij}^* \;=\; \max_{P \in \mathcal{P}_{ij}} \min_{\langle k,l \rangle \in P} A_{kl} \tag{7.2}$$

We are now ready for a trivial but important theorem.

Theorem 7.1. *A pair of pixels is connected in the thresholded affinity graph if and only if their maximin affinity exceeds the threshold value.*

Proof. By definition, a pixel pair is connected in the thresholded affinity graph if and only if there exists a path between them. Such a path is equivalent to a path in the unthresholded affinity graph for which the minimal affinity is above the threshold value. This path in turn exists if and only if the maximin affinity is above the threshold value. □

As a consequence of this theorem, pixel pairs can be classified as connected or disconnected by thresholding maximin affinities. Let $\hat{S}$ be the segmentation produced by thresholding the affinity graph A_{ij} and then finding connected components. Then the connectivity indicator function is

$$\delta(\hat{s}_i, \hat{s}_j) = H(A_{ij}^* - \theta) \tag{7.3}$$

where H is the Heaviside step function.

Maximin affinities can be computed efficiently using minimum spanning tree algorithms Fischer et al. [2004]. A maximum spanning tree is equivalent to a minimum spanning tree, up to a sign change of the weights. Any path in a maximum spanning tree is a maximin path. For our nearest neighbor affinity graphs, the maximin affinity of a pixel pair can be computed in $O(|E| \cdot \alpha(|V|))$ where $|E|$ is the number of graph edges and $|V|$ is the number of pixels and $\alpha(\cdot)$ is the inverse Ackerman function which grows sub-logarithmically. The full matrix A_{ij}^* can be computed in time $O(|V|^2)$ since the computation can be shared. Note that maximin affinities are required for training, but not testing. For segmenting the image at test time, only a connected components computation need be performed, which takes time linear in the number of edges $|E|$.

7.4 Optimizing the Rand index by learning maximin affinities

Since the affinities and maximin affinities are both functions of the image I and the classifier parameters W, we will write them as $A_{ij}(I; W)$ and $A_{ij}^*(I; W)$, respectively. By Eq. (7.3) of

the previous section, the Rand index of Eq. (3.1) takes the form

$$1 - RI(S, I; W) = \binom{N}{2}^{-1} \sum_{i<j} \left| \delta(s_i, s_j) - H(A_{ij}^*(I; W) - \theta) \right|$$

Since this is a discontinuous function of the maximin affinities, we make the usual relaxation by replacing $|\delta(s_i, s_j) - H(A_{ij}^*(I; W) - \theta)|$ with a continuous loss function $l(\delta(s_i, s_j), A_{ij}^*(I; W))$. Any standard loss such as the such as the square loss, $\frac{1}{2}(x - \hat{x})^2$, or the hinge loss can be used for $l(x, \hat{x})$. Thus we obtain a cost function suitable for gradient learning,

$$
\begin{aligned}
E(S, I; W) &= \binom{N}{2}^{-1} \sum_{i<j} l(\delta(s_i, s_j), A_{ij}^*(I; W)) \\
&= \binom{N}{2}^{-1} \sum_{i<j} l(\delta(s_i, s_j), \max_{P \in \mathcal{P}_{ij}} \min_{\langle k,l \rangle \in P} A_{kl}(I; W)) \quad (7.4)
\end{aligned}
$$

The max and min operations are continuous and differentiable (though not continuously differentiable). If the loss function l is smooth, and the affinity $A_{kl}(I; W)$ is a smooth function, then the gradient of the cost function is well-defined, and gradient descent can be used as an optimization method.

Define $(k, l) = mm(i, j)$ to be the maximin edge for the pixel pair (i, j). If there is a tie, choose between the maximin edges at random. Then the cost function takes the form

$$E(S, I; W) = \binom{N}{2}^{-1} \sum_{i<j} l(\delta(s_i, s_j), A_{mm(i,j)}(I; W))$$

It's instructive to compare this with the cost function for standard affinity learning

$$E_{standard}(S, I; W) = \frac{2}{cN} \sum_{\langle i,j \rangle} l(\delta(s_i, s_j), A_{ij}(I; W))$$

where the sum is over all nearest neighbor pixel pairs $\langle i, j \rangle$ and c is the number of nearest neighbors Fowlkes et al. [2003]. In contrast, the sum in the MALIS cost function is over all pairs of pixels, whether or not they are adjacent in the affinity graph. Note that a single

---

Algorithm 7.1 Maximin affinity learning

1. Pick a random pair of (not necessarily nearest neighbor) pixels i and j from a randomly drawn training image I.

2. Find a maximin edge $mm(i,j)$

3. Make the gradient update: $W \leftarrow W + \eta \frac{d}{dW} l(\delta(s_i, s_j), A_{mm(i,j)}(I; W))$

Algorithm 7.2 Standard affinity learning

1. Pick a random pair of nearest neighbor pixels i and j from a randomly drawn training image I

2. Make the gradient update: $W \leftarrow W + \eta \frac{d}{dW} l(\delta(s_i, s_j), A_{ij}(I; W))$

---

edge can be the maximin edge for multiple pairs of pixels, so its affinity can appear multiple times in the MALIS cost function. Roughly speaking, the MALIS cost function is similar to the standard cost function, except that each edge in the affinity graph is weighted by the number of pixel pairs that it causes to be incorrectly classified.

7.5   Online stochastic gradient descent

Computing the cost function or its gradient requires finding the maximin edges for all pixel pairs. Such a batch computation could be used for gradient learning. However, online stochastic gradient learning is often more efficient than batch learning LeCun et al. [1998]. Online learning makes a gradient update of the parameters after each pair of pixels, and is implemented as described in the box.

For comparison, we also show the standard affinity learning Fowlkes et al. [2003]. For each iteration, both learning methods pick a random pair of pixels from a random image. Both compute the gradient of the weight of a single edge in the affinity graph. However, the standard method picks a nearest neighbor pixel pair and trains the affinity of the edge between them. The maximin method picks a pixel pair of arbitrary separation and trains the minimal affinity on a maximin path between them.

Effectively, our connected components performs spatial integration over the nearest neigh-

bor affinity graph to make connectivity decisions about pixel pairs at large distances. MALIS trains these global decisions, while standard affinity learning trains only local decisions. MALIS is superior because it truly learns segmentation, but this superiority comes at a price. The maximin computation requires that on each iteration the affinity graph be computed for the whole image. Therefore it is slower than the standard learning method, which requires only a local affinity prediction for the edge being trained. Thus there is a computational price to be paid for the optimization of a true segmentation error.

7.6 Application to electron microscopic images of neurons

7.6.1 Electron microscopic images of neural tissue

By 3d imaging of brain tissue at sufficiently high resolution, as well as identifying synapses and tracing all axons and dendrites in these images, it is possible in principle to reconstruct connectomes, complete "wiring diagrams" for a brain or piece of brain Seung [2009], Briggman and Denk [2006], Smith [2007]. Axons can be narrower than 100 nm in diameter, necessitating the use of electron microscopy (EM) Seung [2009]. At such high spatial resolution, just one cubic millimeter of brain tissue yields teravoxel scale image sizes. Recent advances in automation are making it possible to collect such images Seung [2009], Briggman and Denk [2006], Smith [2007], but image analysis remains a challenge. Tracing axons and dendrites is a very large-scale image segmentation problem requiring high accuracy. The images used for this study were from the inner plexiform layer of the rabbit retina, and were taken using Serial Block-Face Scanning Electron Microscopy Denk and Horstmann [2004]. Two large image volumes of 100^3 voxels were hand segmented and reserved for training and testing purposes.

7.6.2 Training convolutional networks for affinity classification

Any classifier that is a smooth function of its parameters can be used for maximin affinity learning. We have used convolutional networks (CN), but our method is not restricted to

this choice. Convolutional networks have previously been shown to be effective for similar EM images of brain tissue Jain et al. [2007].

We trained two identical four-layer CNs, one with standard affinity learning and the second with MALIS. The CNs contained 5 feature maps in each layer with sigmoid nonlinearities. All filters in the CN were $5 \times 5 \times 5$ in size. This led to an affinity classifier that uses a $17 \times 17 \times 17$ cubic image patch to classify a affinity edge. We used the square-square loss function $l(x, \hat{x}) = x \cdot \max(0, 1 - \hat{x} - m)^2 + (1 - x) \cdot \max(0, \hat{x} - m)^2$, with a margin $m = 0.3$.

As noted earlier, maximin affinity learning can be significantly slower than standard affinity learning, due to the need for computing the entire affinity graph on each iteration, while standard affinity training need only predict the weight of a single edge in the graph. For this reason, we constructed a proxy training image dataset by picking all possible $21 \times 21 \times 21$ sized overlapping sub-images from the original training set. Since each $21 \times 21 \times 21$ sub-image is smaller than the original image, the size of the affinity graph needed to be predicted for the sub-image is significantly smaller, leading to faster training. A consequence of this approximation is that the maximum separation between image pixel pairs chosen for training is less than about 20 pixels. A second means of speeding up the maximin procedure is by pretraining the maximin CN for 500,000 iterations using the fast standard affinity classification cost function. At the end, both CNs were trained for a total of 1,000,000 iterations by which point the training error plateaued.

7.6.3 Maximin learning leads to dramatic improvement in segmentation performance
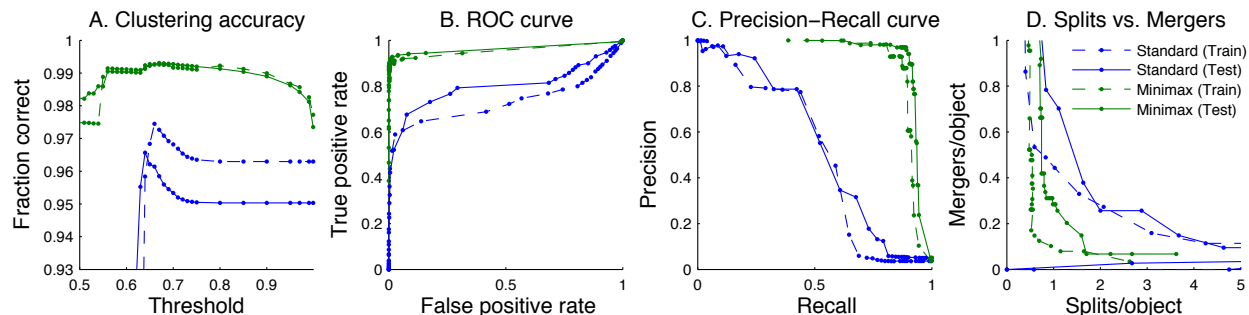
Figure 7-2: **Quantification of segmentation performance on 3d electron microscopic images of neural tissue.** A) Clustering accuracy measuring the number of correctly classified pixel pairs. B) and C) ROC curve and precision-recall quantification of pixel-pair connectivity classification shows near perfect performance. D) Segmentation error as measured by the number of splits and mergers.

We benchmarked the performance of the standard and maximin affinity classifiers by measuring the the pixel-pair connectivity classification performance using the Rand index. After training the standard and MALIS affinity classifiers, we generated affinity graphs for the training and test images. In principle, the training algorithm suggests a single threshold for the graph partitioning. In practice, one can generate a full spectrum of segmentations leading from over-segmentations to under-segmentations by varying the threshold parameter. In Fig. 7-2, we plot the Rand index for segmentations resulting from a range of threshold values.

In images with large numbers of segments, most pixel pairs will be disconnected from one another leading to a large imbalancing the number of connected and disconnected pixel pairs. This is reflected in the fact that the Rand index is over 95% for both segmentation algorithms. While this imbalance between positive and negative examples is not a significant problem for training the affinity classifier, it can make comparisons between classifiers difficult to interpret. Instead, we can use the ROC and precision-recall methodologies, which provide for accurate quantification of the accuracy of classifiers even in the presence of large class imbalance. From these curves, we observe that our maximin affinity classifier dramatically outperforms the standard affinity classifier.

Our positive results have an intriguing interpretation. The poor performance of the con-

| image | groundtruth | maximin training | standard training |

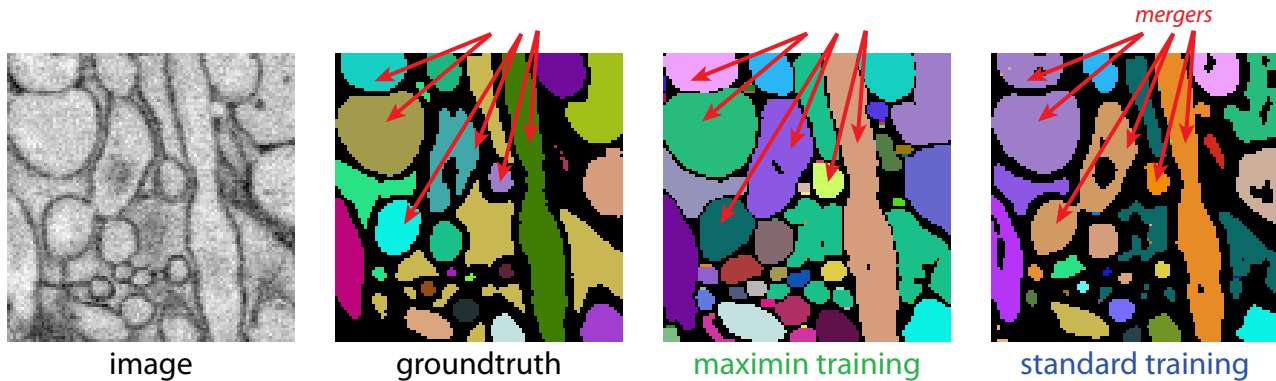Figure 7-3: **A 2d cross-section through a 3d segmentation of the test image.** The maximin segmentation correctly segments several objects which are merged in the standard segmentation, and even correctly segments objects which are missing in the groundtruth segmentation. Not all segments merged in the standard segmentation are merged at locations visible in this cross section. Pixels colored black in the machine segmentations correspond to pixels completely disconnected from their neighbors and represent boundary regions.

nected components when applied to a standard learned affinity classifier could be interpreted to imply that 1) a local classifier lacks the context important for good affinity prediction; 2) connected components is a poor strategy for image segmentation since mistakes in the affinity prediction of just a few edges can merge or split segments. On the contrary, our experiments suggest that when trained properly, thresholded affinity classification followed by connected components can be an extremely competitive method of image segmentations.

7.7 Discussion

In this paper, we have trained an affinity classifier to produce affinity graphs that result in excellent segmentations when partitioned by the simple graph partitioning algorithm of thresholding followed by connected components. The key to good performance is the training of a segmentation-based cost function, and the use of a powerful trainable classifier to predict affinity graphs. Once trained, our segmentation algorithm is fast. In contrast to classic graph-based segmentation algorithms where the partitioning phase dominates, our partitioning algorithm is simple and can partition graphs in time linearly proportional to the number of edges in the graph. We also do not require any prior knowledge of the number of image segments or image segment sizes at test time, in contrast to other graph partitioning

algorithms [Felzenszwalb and Huttenlocher, 2004, Shi and Malik, 2000].

The formalism of maximin affinities used to derive our learning algorithm has connections to single-linkage hierarchical clustering, minimum spanning trees and ultrametric distances. Felzenszwalb and Huttenlocher [2004] describe a graph partitioning algorithm based on a minimum spanning tree computation which resembles our segmentation algorithm, in part. The Ultrametric Contour Map algorithm [Arbelaez, 2006] generates hierarchical segmentations nearly identical those generated by varying the threshold of our graph partitioning algorithm. Neither of these methods incorporates a means for learning from labeled data, but our work shows how the performance of these algorithms can be improved by use of our maximin affinity learning.

Chapter 8

Concluding thoughts

8.1 A method for neural circuit reconstruction

In this thesis, I presented a new method for the challenging problem of automatic recon-
struction of neural circuits from 3d electron micrographs. This method is *adaptive* it uses
machine learning and groundtruth human reconstructions of small volumes of imagery to
specialize to the unique properties (staining, imaging, cell types) of a particular dataset but
also to *generalize* to imagery of the same type previously unseen. It is expected that such
an adaptive method can be tuned to outperform other engineered non-adaptive methods. I
believe this method to be the best currently available for this task. However, performance is
not the only consideration when dealing with images that can be hundreds of terabytes large;
speed is of equal consideration. The method presented here can generate segmentations in
linear time in the size of the image. This is the best possible asymptotic performance, since
every pixel in the image must atleast be examined once. Thus this method is provably the
fastest.

Is this method good enough? No. Current error rates are still on the order of one mistake
per tens of microns of neurite. However, error rates on the order of a mistake per millimeter
or better are probably achievable through the training of convolutional networks with larger
fields of view and more free parameters, in conjunction with better model selection and
cross validation. The primary obstacle to this is the time required to train the convolutional
networks, which currently takes on the order of months. It is expected that the use of

modern computing architectures such as GPUs will yield speedups of several-fold to an order of magnitude or more. Clusters of such computers can be used to perform a parallel search over the space of convolutional network architectures. It is also conceivable that recent semi-supervised training methods [Hinton and Salakhutdinov, 2006, Bengio et al., 2007] will lead to improvements in training time and generalization performance.

Ultimately, it is possible that the method described in this thesis will saturate and radically different methods will be required to assemble fragments generated by this method in conjunction with a high-level model of neuronal shape. As with speech analysis, there will likely be low-level, mid-level and high-level models which together will yield the performance required for the success of connectomics.

8.2   Measures for comparing reconstructions

With the rapid growth of research into automated reconstruction methods, it is crucial to have good measures of the quality of reconstructions for comparison purposes. It is hoped that the methods presented here will find good use.

8.3   Convolutional networks for computing hierarchical segmentations

At first glance, convolutional networks appear as a curious choice for image segmentation, given their origin in application to high-level vision tasks such as digit and object recognition. In this thesis, I show that they are uniquely suited to image processing applications such as boundary detection. While boundary detection is usually considered a low-level vision task prone to errors and mis-interpretation, with the use of convolutional networks, we can achieve low-level boundary decisions that are informed by a powerful architecture capable of using high-level information to correctly assign low-level boundaries.

It is generally assumed that low-level boundary detectors cannot help but produce mistakes such as numerous tiny breaks in the boundaries. This has lead to the development of models with strong priors for local continuity such as markov random fields and level sets.

However, producing locally closed contours is a rather trivial task for more modern classifiers. What was truly lacking was a good means of training the classifier to produce closed contours. This problem has been remedied by the ultrametric learning procedure introduced in this thesis.

Probabilistic random fields, graph partitioning and level set based methods currently constitute the state of the art in computer vision for image segmentation. In their canonical form, the success of all of these methods is due in large part to the use of strong low-level priors to exclude segmentations that are really bad. In the absence of large amounts of training data, such methods are the only reasonable methods. However, in the large training data limit, the effect of a prior is minimized and it is important to have flexible and powerful discriminitively trained methods. In principle, random field and graph partitioning models can be trained discriminitively using the methods provided in this thesis. But it is still likely that convolutional networks with many hundreds of thousands of free parameters will outperform other models.

While the application of machine learning for image segmentation in this thesis has been restricted to 3d EM images of brain tissue, the methods are not restricted to connectomics. They make no assumptions regarding the type of image to be segmented and can be used directly with no modification to any image domain with training data.

8.4 Generalizing ultrametric learning

The learning algorithm for image segmentation relied on learning a special kind of distance between image pixels known as an ultrametric distance. It is a specialization of regular metric distances with rather curious and useful properties. The most interesting property is the correspondence of ultrametric distances to dendrogram representations of hierarchical clustering. In this thesis, I presented a means for learning minimax distances corresponding to single-linkage clustering.

While there has been some work in learning hierarchies and tree structured representations of data, I believe the minimax learning algorithm to be uniquely useful. In contrast to other methods, single-linkage clustering can be performed exactly in polynomial time.

Further, our training of this ultrametric requires little in the way of approximations. I hope this model of hierarchy learning will find more general utility than image segmentation.

I believe ultrametric learning in its most general setting will prove to be a promising avenue for future research.

Bibliography

K. A. Al-Kofahi, S. Lasek, D. H. Szarowski, C. J. Pace, G. Nagy, J. N. Turner, and B. Roysam. Rapid automated three-dimensional tracing of neurons from confocal image stacks. *Information Technology in Biomedicine, IEEE Transactions on*, 6(2):171–187, 2002.

B. Andres, U. Kothe, M. Helmstaedter, W. Denk, and F.A. Hamprecht. Segmentation of SBFSEM Volume Data of Neural Tissue by Hierarchical Classification. *Lecture Notes in Computer Science*, 5096:142–152, 2008.

P. Arbelaez. Boundary extraction in natural images using ultrametric contour maps. *Proc. POCV*, 2006.

F.R. Bach and M.I. Jordan. Learning spectral clustering, with application to speech separation. *The Journal of Machine Learning Research*, 7:1963–2001, 2006.

Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, and U. Montreal. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*, page 153. The MIT Press, 2007.

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

Kevin L Briggman and Winfried Denk. Towards neural circuit reconstruction with volume electron microscopy techniques. *Curr Opin Neurobiol*, 16(5):562–70, 2006. ISSN 0959-4388 (Print).

I Carlbom, D Terzopoulos, and K M Harris. Computer-assisted registration, segmentation, and 3d reconstruction from images of neuronal tissue sections. *IEEE Trans Med Imaging*, 13(2):351–362, 1994. ISSN 0278-0062 (Print). doi: 10.1109/42.293928.

B. Chazelle. A minimum spanning tree algorithm with inverse-Ackermann type complexity. *Journal of the ACM*, 47(6):1028–1047, 2000.

D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, pages 603–619, 2002. URL http://doi.ieeecs.org/10.1109/34.1000236.

T.H. Cormen, C.E. Leiserson, and R.L. Rivest. Introduction to algorithms, 1990.

T. Cour, F. Benezit, and J. Shi. Spectral Segmentation with Multiscale Graph Decomposition. *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)-Volume 2-Volume 02*, pages 1124–1131, 2005a.

Timothee Cour, Nicolas Gogin, and Jianbo Shi. Learning spectral graph segmentation. In *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics*, 2005b.

Winfried Denk and Heinz Horstmann. Serial block-face scanning electron microscopy to reconstruct three-dimensional tissue nanostructure. *PLoS Biol*, 2(11):e329, 2004. ISSN 1545-7885 (Electronic).

Inderjit Dhillon, Yuqiang Guan, and Brian Kulis. A fast kernel-based multilevel algorithm for graph clustering. In *KDD '05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 629–634, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-135-X. doi: http://doi.acm.org/10.1145/1081870.1081948.

P. Dollár, Z. Tu, and S. Belongie. Supervised learning of edges and object boundaries. In *CVPR*, June 2006.

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

P.F. Felzenszwalb and D.P. Huttenlocher. Efficient Graph-Based Image Segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

J C Fiala. Reconstruct: a free editor for serial section microscopy. *J Microsc*, 218(Pt 1): 52–61, Apr 2005. ISSN 0022-2720 (Print). doi: 10.1111/j.1365-2818.2005.01466.x.

B. Fischer, V. Roth, and J.M. Buhmann. Clustering with the connectivity kernel. In *Advances in Neural Information Processing Systems 16: Proceedings of the 2003 Conference*. Bradford Book, 2004. URL `http://books.google.com/books?hl=en&lr=&id=0F-9C7K8fQ8C&oi=fnd&pg=PA89&dq=connectivity+kernel&ots=TFFzm0T693&sig=OB4djRrt91kr_CH0tARk0-fP7P0`.

C. Fowlkes, D. Martin, and J. Malik. Learning affinity functions for image segmentation: combining patch-based and gradient-based approaches. *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, 2, 2003.

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

R.C. Gonzalez and R.E. Woods. Digital image processing, 2002.

JA Hartigan and MA Wong. A k-means clustering algorithm. *JR Stat. Soc. Ser. C-Appl. Stat*, 28:100–108, 1979.

KJ Hayworth, N. Kasthuri, R. Schalek, and JW Lichtman. Automating the collection of ultrathin serial sections for large volume TEM reconstructions. *Microscopy and Microanalysis*, 12(S02):86–87, 2006.

X. He, R. Zemel, and M. Carreira-Perpinan. Multiscale conditional random fields for image labeling. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2. IEEE Computer Society; 1999, 2004.

Moritz Helmstaedter, Kevin L Briggman, and Winfried Denk. 3d structural imaging of the brain with photons and electrons. *Curr Opin Neurobiol*, 18(6):633–641, 2008. ISSN 1873-6882 (Electronic). doi: 10.1016/j.conb.2009.03.005.

GE Hinton and RR Salakhutdinov. Reducing the dimensionality of data with neural networks, 2006.

V. Jain, J.F. Murray, F. Roth, S. Turaga, V. Zhigulin, K.L. Briggman, M.N. Helmstaedter, W. Denk, and H.S. Seung. Supervised learning of image restoration with convolutional networks. *ICCV 2007*, 2007.

Elizabeth Jurrus, Tolga Tasdizen, Pavel Koshevoy, P. Thomas Fletcher, Melissa Hardy, Chi-Bin Chien, Winfried Denk, and Ross Whitaker. Axon tracking in serial block-face scanning electron microscopy. In *Workshop on Microscopic Image Analysis with Applications in Biology*, 2006.

S. Kumar and M. Hebert. Discriminative random fields: a discriminative framework for contextual interaction in classification. *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1150–1157, 2003.

Y. LeCun, B. Boser, JS Denker, D. Henderson, RE Howard, W. Hubbard, and LD Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4): 541–551, 1989.

Y. LeCun, L. Bottou, G.B. Orr, and K.R. Müller. Efficient backprop. *Lecture notes in computer science*, pages 9–50, 1998.

M. Maire, P. Arbelaez, C. Fowlkes, and J. Malik. Using contours to detect and localize junctions in natural images. In *IEEE Conference on Computer Vision and Pattern Recognition, 2008. CVPR 2008*, pages 1–8, 2008.

D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. Eighth Int'l Conf. Computer Vision*, volume 2, pages 416–423, 2001.

David R Martin, Charless C Fowlkes, and Jitendra Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Trans Pattern Anal Mach Intell*, 26(5):530–549, May 2004. ISSN 0162-8828 (Print). doi: 10.1109/TPAMI. 2004.1273918.

M. Meila and J. Shi. Learning segmentation by random walks. *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, pages 873–879, 2001. URL `http://www.stat.washington.edu/mmp/Papers/nips2000.ps`.

F. Meyer. Topographic distance and watershed lines. *Signal Processing*, 38(1):113–125, 1994.

Yuriy Mishchenko. Automation of 3d reconstruction of neural tissue from large volume of conventional serial section transmission electron micrographs. *J Neurosci Methods*, 176 (2):276–289, Jan 2009. ISSN 0165-0270 (Print). doi: 10.1016/j.jneumeth.2008.09.006.

Feng Ning, Damien Delhomme, Yann LeCun, Fabio Piano, Leon Bottou, and Paolo Barbano. Toward automatic phenotyping of developing embryos from videos. *IEEE Transactions on Image Processing*, 14(9):1360–1371, September 2005. Special issue on Molecular and Cellular Bioimaging.

W.M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, pages 846–850, 1971.

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

H.S. Seung. Reading the Book of Memory: Sparse Sampling versus Dense Mapping of Connectomes. *Neuron*, 62(1):17–29, 2009.

Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000. URL `citeseer.csail.mit.edu/article/shi97normalized.html`.

C. Sinthanayothin, J.F. Boyce, H.L. Cook, and T.H. Williamson. Automated localisation of the optic disc, fovea, and retinal blood vessels from digital colour fundus images, 1999.

Stephen J Smith. Circuit reconstruction tools today. *Curr Opin Neurobiol*, 17(5):601–608, Oct 2007. ISSN 0959-4388 (Print). doi: 10.1016/j.conb.2007.11.004.

Z. Tu, X. Chen, A.L. Yuille, and S.C. Zhu. Image parsing: Unifying segmentation, detection, and recognition. *International Journal of Computer Vision*, 63(2):113–140, 2005.

S.C. Turaga, J.F. Murray, V. Jain, F. Roth, M. Helmstaedter, K. Briggman, W. Denk, and H.S. Seung. Convolutional networks can learn to generate affinity graphs for image segmentation. *Neural Computation*, in press.

S.C. Turaga, K. Briggman, M. Helmstaedter, W. Denk, and H.S. Seung. Maximin affinity learning of image segmentation. *NIPS 2009*, submitted.

R. Unnikrishnan, C. Pantofaru, and M. Hebert. Toward objective evaluation of image segmentation algorithms. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, pages 929–944, 2007. URL `http://doi.ieeecomputersociety.org/10.1109/TPAMI.2007.1046`.

L. Vincent and P. Soille. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 13 (6):583–598, 1991. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=87344`.

J. G. White, E. Southgate, J. N. Thomson, and S. Brenner. 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, nov 1986. ISSN 0080-4622.