Machine Learning and Optimization for Neural Circuit Reconstruction

Jeremy Maitin-Shepard

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2015-239
http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-239.html

December 16, 2015

Machine Learning and Optimization for Neural Circuit Reconstruction

by

Jeremy Bertram Maitin-Shepard

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Associate Professor Pieter Abbeel, Chair
Professor Jitendra Malik
Professor Bruno Olshausen

Fall 2015

Machine Learning and Optimization for Neural Circuit Reconstruction

Abstract

Machine Learning and Optimization for Neural Circuit Reconstruction

by

Jeremy Bertram Maitin-Shepard

Doctor of Philosophy in Computer Science

University of California, Berkeley

Associate Professor Pieter Abbeel, Chair

Mapping neuroanatomy, in the pursuit of linking hypothesized computational models consistent with observed functions to the actual physical structures, has been a long-standing fundamental problem in neuroscience. One primary interest is in mapping the network structure of neural circuits by identifying the morphology of each neuron and the locations of synaptic connections between neurons, a field of study called connectomics. Currently, the most promising approach for obtaining such maps of neural circuit structure is volume electron microscopy of a stained and fixed block of tissue.

While recent advances in volume electron microscopy make feasible the imaging of very large circuits at sufficient resolution to discern even the smallest neuronal processes, image analysis remains a key challenge limiting the rate of discovery. Existing fully-automated algorithms offer inadequate accuracy to replace human annotators, and semi-automated methods offer only limited speedup. Towards addressing this image analysis problem, we designed, implemented, and evaluated novel methods based on machine learning and optimization related to three different sub-problems:

Detection of cell boundaries at the per-voxel level is a key analysis step, given that cell boundaries serve as the primary indication of cell morphology, We propose a highly-scalable, layered architecture for classification on 3-D volumes: unlike conventional dense deep learning approaches, this architecture relies on simple, parallelizable clustering algorithms and convex optimization to learn wide, sparse models. By exploiting rotational invariance of the data distribution and a highly-efficient distributed GPU implementation, we achieved performance comparable to or better than deep convolutional networks trained for weeks with only several hours of training, enabling much faster iteration on model design.

Certain promising high-throughput microscopy techniques result in significant discontinuities between section images even after alignment, due to variations in imaging conditions and section thickness, among other artifacts. These artifacts impede truly 3-D analysis of these volumes. We propose an iterative coarse-to-fine procedure that optimizes the parameters of spatially vary linear transformations of the intensity data in order to minimize discontinuities along the section axis, subject to detail-preserving regularization. Testing showed this technique to yield significant quantitative improvement in image quality, and qualitatively corrected essentially all visible discontinuities without any noticeable loss of detail; it also significantly improved 3-D segmentation accuracy.

To integrate higher-level prior information about shape, we introduce a new machine learning approach for image segmentation, based on a joint energy model over image features and novel *local binary shape descriptors*. These descriptors compactly represent rich shape information at multiple scales, including *interactions between* multiple objects. Our approach reflects the inherent combinatorial nature of dense image segmentation problems. We propose efficient algorithms for learning deep neural networks to model the joint energy, and for local optimization of this energy in the space of supervoxel agglomerations. This architecture yields state-of-the-art performance on several challenging electron microscopy datasets.

These advances constitute critical progress towards fully-automated reconstruction of circuits of hundreds of thousands of neurons.

To Sister Theresa Munger, who through her kindness, generosity, dedication and humor made the world a better place.

Contents

List of Figures

List of Tables

List of Algorithms

Glossary

$A^t[C]$ the *active set* of actions at step t that are supervoxel merges in $S^t[C]$. 70

$A^t_Z[C]$ the set of actions at step t that are active in zone Z of $S^t[C]$. 76

A^t the set of remaining merge actions after t steps of agglomeration. 67

G/T the contraction of G by each set of vertices in T, where T is a partition of vertices(G).

$G[V']$ the subgraph of G induced by the vertex subset V'. 61

K the set of vertices that make up a connected component of an undirected graph.

$S + e$ the segmentation that results from adding the edge set e to the voxel graph S. 63

S^t the segmentation after t steps of agglomeration. 65

S a segmentation, represented as an undirected graph over voxels. 58

$\mathbb{Z}_{>0}$ the set of strictly positive integers.

$\mathbb{Z}$ the set of integers.

$\hat{V}_s(x; S)$ the approximate component visibility set, the set of connected components at positions within a bounding box of size B_s around x. 73

$|X|$ the cardinality of the set X.

$\mathcal{K}(G)$ the set of connected components of the undirected graph G. 61

$\mathcal{C}_s(x)$ the connectivity region used for computing the type s shape descriptor at position x. 61

X^s_C the set of (type s) shape descriptor center positions for which the descriptor bounding box is contained within C. 59

$\bar{B}_s$ the maximum dimensions, as a vector in $\mathbb{Z}^3_{>0}$, of a connectivity region used for computing shape descriptors of type s. 59

$\mathcal{C}_s$ the set of connectivity regions obtained as regular overlapping tiles of size $\bar{B}_s$ and stride stride_s. 59

stride_s the stride, as a vector in $\mathbb{Z}^3_{>0}$, at which overlapping connectivity regions are placed for computing shape descriptors of type s. 59

$r_s(x; S)$ the type s local binary shape descriptor centered at position x in the segmentation S. 61

$\text{edges}(G)$ the edge set of the graph G.

$\Delta_S^{+e,+e'} f(S)$ the forward second-order discrete derivative of f with respect to S. 64

$\Delta_S^{+e} f(S)$ the forward discrete derivative of f with respect to S. 64

$\phi(x; I)$ a feature representation of the image context centered around position x. 62

$E(S; I)$ the global energy representing the compatibility between a segmentation S and corresponding image I. 62

$\mathbb{1}[x]$ indicator variable equal to 1 if condition x holds, or 0 otherwise.

$[X]^k$ the set of k-subsets of X, i.e. $[X]^k \coloneqq \{Y \subseteq X \mid |Y| = k\}$.

$E_s(x; S; I)$ the type s local energy term representing the compatibility at position x between a segmentation S and corresponding image I. 62

$\hat{E}_s(r; v)$ the local energy as a function of the type s shape descriptor r and corresponding image feature vector v. 62

$\mathcal{N}(x)$ the von Neumann neighborhood of x. 59

$\odot$ the pointwise product. 73

2^X the power set of X, i.e. the set of all subsets of X.

$R_{\vec{a}}^{\vec{b}}$ the hyperrectangle of points x satisfying $\vec{a} \le x < b$. 73

B_s shape descriptor bounding box dimensions vector in $\mathbb{Z}^3$. 55

$\text{vertices}(G)$ the vertex set of the graph G.

$V_s(x; S)$ component visibility set, the set of connected components at positions sampled around x by the shape descriptor s. 61

$W_s(Z; S)$ component visibility set for a zone Z, the set of connected components at positions sampled around any position $x \in Z$ by the shape descriptor s. 75

$W_s^{-1}(K; C)$ zone visibility set. 75

$\mathcal{Z}_{s,C}$ the set of zones that partition X_C^s. 75

$e[C]$ the subset of edge set e restricted to vertices in the vertex set C. 67

$e[S]$ the subset of edge set e restricted to vertices of graph S. 67

$e_{K,K'}$ the set of edges between neighboring voxels between components K and K'. 67

r binary shape descriptor vector. 57

s binary shape descriptor specification. 55

neurite an axon or dendrite projecting from the cell body of a neuron.

redundant merge a set of edges that all correspond to a self edge in the graph obtained by contracting connected components. 68

spanning subgraph a subgraph with the same vertex set.

supervoxel merge a set of edges that all correspond to a single edge in the graph obtained by contracting connected components. 68

Chapter 1

Introduction

Mapping neuroanatomy, in the pursuit of linking hypothesized computational models consistent with observed functions to the actual physical structures, has been a long-standing fundamental problem in neuroscience. One primary interest is in mapping the network structure of neural circuits by identifying the morphology of each neuron and the locations of synaptic connections between neurons, a field of study called connectomics. Currently, the most promising approach for obtaining such maps of neural circuit structure is volume electron microscopy of a stained and fixed block of tissue. [17, 42, 43, 31] This technique was first used successfully decades ago in mapping the structure of the complete nervous system of the 302-neuron *Caenorhabditis elegans*; due to the need to manually cut, image, align and trace all neuronal processes in about 8000 50 nm serial sections, even this small circuit required over 10 years of labor, much of it spent on image analysis. [86] At the time, scaling this approach to larger circuits did not appear practical.

Recent advances in volume electron microscopy [32, 59, 40] make feasible the imaging of large circuits, potentially containing hundreds of thousands of neurons, at sufficient resolution to discern even the smallest neuronal processes. [17, 42, 43, 31] The high image quality and near-isotropic resolution achievable with these methods enables the resultant data to be treated as a true 3-D volume, which significantly aids reconstruction of processes that do not run parallel to the sectioning axis, and is potentially more amenable to automated image processing.

Image analysis remains a key challenge, however. The primary bottleneck is in segmenting the full volume, which is filled almost entirely by heavily intertwined neurons, into the volumes occupied by each individual neuron. While the cell boundaries shown by the stain provide a strong visual cue in most cases, neurons can extend for tens of centimeters in path length while in some places becoming as narrow as 40 nm; a single mistake anywhere along the path can render connectivity information for the neuron largely inaccurate. Existing automated and semi-automated segmentation methods do not sufficiently reduce the amount of human labor required. A recent reconstruction of 950 neurons in the mouse retina required over 20000 hours of human labor, even with an efficient method of tracing just a skeleton of each neuron [44]; a recent reconstruction of 379 neurons in the *Drosophila* medulla column (part of the visual pathway) required 12940 hours of manual proof-reading/correction of an automated segmentation [79].

This thesis gives an overview of the field and addresses several key aspects of this image analysis problem:

As background, in chapter 2, we give an overview of volume electron microscopy imaging techniques. In chapter 3, we discuss rotational invariance/covariance, boundary representation, and prior work related to automated circuit reconstruction from electron microscopy volumes. In chapter 4, we discuss metrics for measuring segmentation accuracy.

Summary of Contributions

In chapter 5, we propose an alternative to deep neural networks for boundary classification based instead on highly-parallel sparse, wide networks. These models matched or exceeded the accuracy of deep neural network-based boundary classifiers, with only a fraction of the training time.

In chapter 6, we address the problem that certain promising high-throughput microscopy techniques result in significant discontinuities between section images even after alignment, due to variations in imaging conditions and section thickness, among other artifacts. These artifacts impede truly 3-D analysis of these volumes. We propose an iterative coarse-to-fine procedure that optimizes the parameters of spatially vary linear transformations of the intensity data in order to minimize discontinuities along the section axis, subject to detail-preserving regularization. As we previously published [6], testing showed this technique to yield significant quantitative improvement in image quality, and qualitatively corrected essentially all visible discontinuities without any noticeable loss of detail; it also significantly improved 3-D segmentation accuracy.

In chapter 7, we describe a new approach for image segmentation based on a learned joint energy model representing the compatibility between the image and a candidate segmentation, using a novel binary shape descriptor for representing local configurations of 3-D objects. These descriptors compactly represent rich shape information at multiple scales, including *interactions between* multiple objects. Our approach reflects the inherent combinatorial nature of dense image segmentation problems. We propose efficient algorithms for learning deep neural networks to model the joint energy, and for local optimization of this energy in the space of supervoxel agglomerations. We demonstrate state-of-the-art performance on several challenging electron microscopy datasets.

Chapter 2

Volume Electron Microscopy Methods

Because electrons cannot penetrate tissue more than a few tens of nanometers, volume electron microscopy is based on one of two approaches: *serial section imaging*, in which physical sections are cut using an ultramicrotome prior to imaging, or *block face imaging*, based an alternating process of imaging the top surface of a block of tissue, then scraping or ablating the top surface to expose a lower layer, as shown in fig. 2.1. Significant advances in these methods have been made in recent years, but there are still important trade-offs that must be made. We briefly describe these methods and trade-offs here; refer to the review by Briggman and Bock [16] for a more comprehensive discussion. The sample preparation, another active area of research, is not discussed here, but critically affects contrast in the images, and also can determine such factors as the extent to which extracellular space is expanded or compressed.

2.1 Serial section imaging

Serial Section Transmission Electron Microscopy (ssTEM)

Serial section imaging is the basis for the earliest approach to volume electron microscopy, serial section Transmission Electron Microscopy (ssTEM), which was used to reconstruct the nervous system of *Caenorhabditis elegans*. Sections as thin as 40 nm are cut using an ultramicrotome, manually placed on a support grid, and imaged using transmission electron microscopy, which offers high imaging throughput. The cut sections can be imaged independently, such that the overall acquisition speed is not limited by the speed of a single microscope. They can also be preserved, in order to be reimaged if an imaging problem is discovered later, and can also potentially be imaged using optical microscopy as well, and subjected to multiple cycles of staining and rinsing, in order to obtain additional information. The thinness of the sections is limited, however, by the fragility of the tissue, which is supported only by the mesh support grid, and there is a risk of tissue distortion and damage.

The depth resolution can be improved by the use of electron tomography techniques,

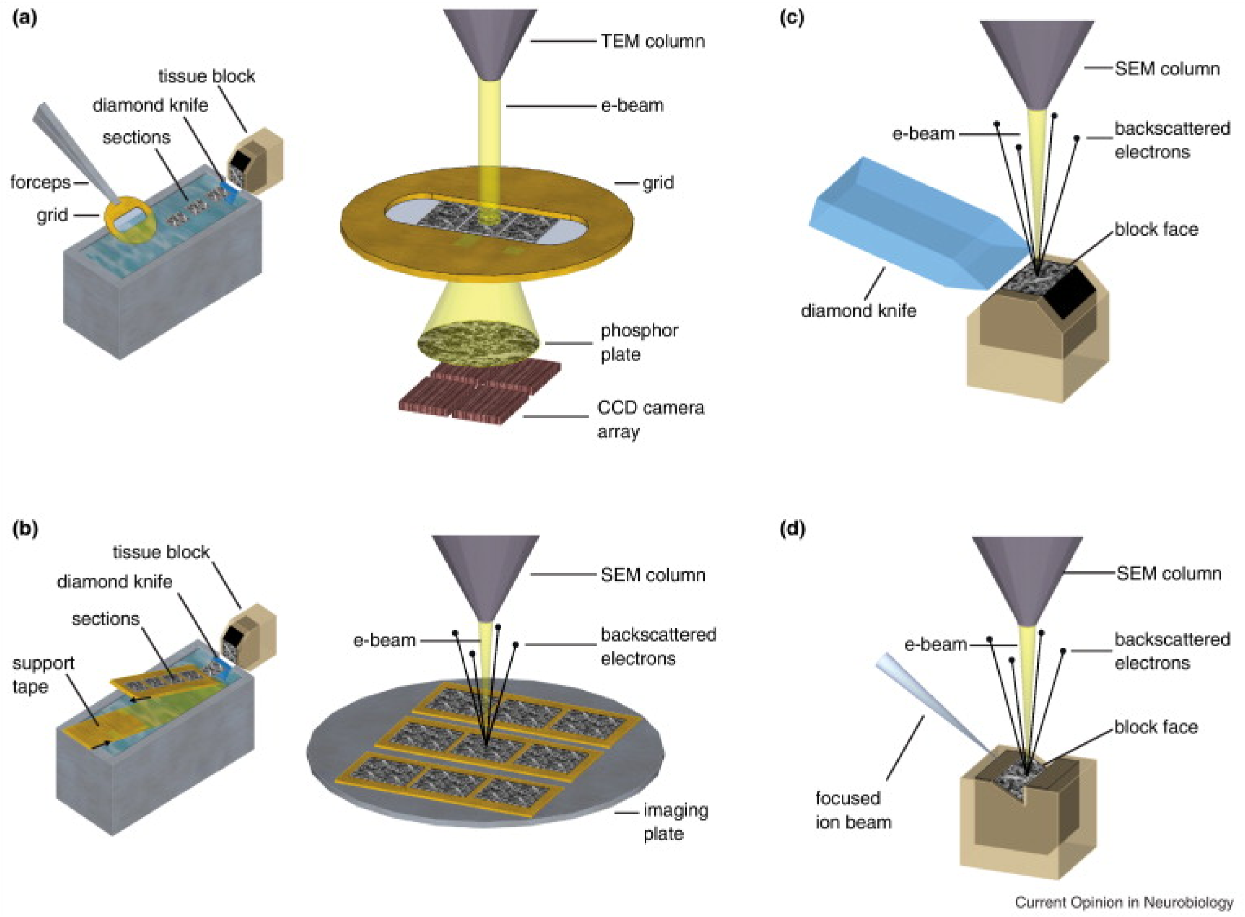


Figure 2.1: Volume electron microscopy methods. The two serial section imaging methods are shown on the left: (a) Serial section Transmission Electron Microscopy (ssTEM) and (b) Automated Tape-collection Ultramicrotome Scanning Electron Microscopy (ATUM-SEM). The two blockface imaging are shown on the right: (c) Serial Block Face Scanning Electron Microscopy (SBEM) and (d) Focused Ion Beam Scanning Electron Microscopy (FIB-SEM). Figure by Kevin Briggman and Davi Bock, 2012, reprinted with permission from Elsevier. [16]

in which sections are imaged at many angles in order to obtain a computed 3-D view. Acquisition rate is severely limited, however, by the requirement to take many images per section. There has been work, however, on using machine learning to approximate a high resolution 3-D result using fewer images.

Distortion and the independent imaging of sections does limit the ability to accurately align sections at full resolution in order to obtain what can be considered a true 3-D volume (rather than a 2.5D stack of sections).

Automated Tape-collection Ultramicrotome Scanning Electron Microscopy (ATUM-SEM)

The recent technique of Automated Tape-collection Ultramicrotome Scanning Electron Microscopy (ATUM-SEM) [75] replaces the manual section handling with a support tape that picks up sections automatically as they are cut. The automated section cutting saves labor and improves reliability, and the backing tape allows sections as thin as 25 nm to be used reliably. The backing tape preserves the sections well, and as with ssTEM the acquisition speed is not limited by the speed of a single microscope.

Because the backing tape is electron opaque, Scanning Electron Microscopy (SEM) in backscatter or secondary electron detection mode must be used. While previously this slowed throughput relative to Transmission Electron Microscopy, recently developed multi-beam scanning electron microscopes have largely closed this gap.

The backing tape prevents the use of traditional electron tomography techniques, but a recently developed deconvolution technique based on Scanning Electron Microscopy at multiple energies can be used instead to obtain higher effective depth resolution.

2.2 Block face imaging

Serial Block Face Scanning Electron Microscopy (SBEM)

Serial Block Face Scanning Electron Microscopy (SBEM) [32] dispenses with preserved sections and instead uses a repeated process of:

1. imaging the top surface of a block of tissue using Scanning Electron Microscopy;
2. advancing the stage upwards;
3. scraping off a thin layer of tissue using a diamond knife.

No longer constrained by the need to maintain the physical integrity of sections, the tissue depth scraped off between consecutive images can be thin as 20 nm. The entire process of cutting an imaging of a block of tissue occurs inside the hard vacuum of the microscope. This process naturally leads to excellent alignment between section images, and minimizes tissue distortion and damage. Acquisition speed is limited, however, by the speed of a single microscope, and because it is a destructive process, there is no possibility of re-imaging a section if a problem is discovered later.

Multi-energy imaging-based deconvolution can be used in conjunction with this method to further improve the depth resolution.

Focused Ion Beam Scanning Electron Microscopy (FIB-SEM)

Focused Ion Beam Scanning Electron Microscopy (FIB-SEM) [59] is variant of this method that replaces the diamond knife with a focused beam of ions that ablate the top surface of the tissue. This allows depth resolution as low as 2 nm. The lateral field of view possible with current FIB-SEM microscopes is severely limited, however.

A recently demonstrated “hot knife” technique [39] potentially both mitigates the limits on field of view and also allows parallel imaging using multiple microscopes, for both diamond knife and Focused Ion Beam-based block face imaging.

Chapter 3

Prior work

Existing work has addressed the computational problem of automated object reconstruction from electron microscopy volumes in diverse ways, including slice-to-slice contour tracking, boundary classification based on various machine learning methods, and agglomeration based on various global consistency constraints and various learned policies. Aside from slice-to-slice contour tracking, these methods are variants of a common pipeline shown in figs. 3.1 and 3.2.

While this reconstruction problem is in principle quite related to the conventional computer vision problems of image segmentation and edge detection for natural scenes, in practice the challenges are fundamentally quite different. Segmentation of natural scenes is complicated by 2-D projection, occlusion and lighting effects, but is made easier by strong local appearance cues for distinguishing segments, and more rigid geometry. Neural circuit reconstruction also requires very high accuracy, due to the tendency for false merge errors to propagate over the extremely large volumes that these circuits span, but the variation in the data is fairly low. In contrast, natural scenes tend to have significant inherent ambiguity,

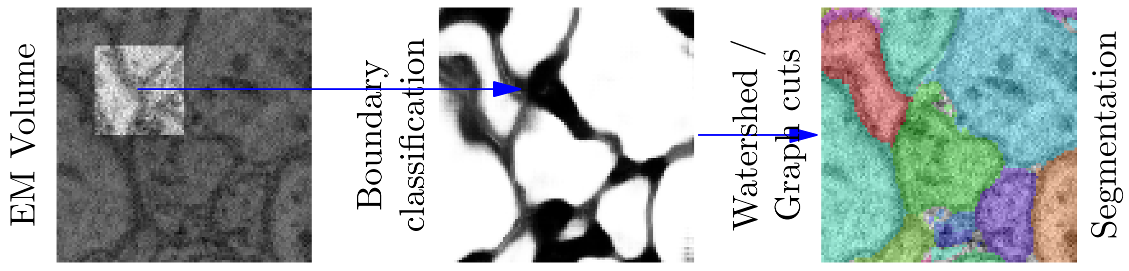


Figure 3.1: Architecture of prior methods based on boundary classification. A boundary detection step establishes local hypotheses of object boundaries, typically by applying a learned classifier in a sliding window fashion, as described in section 3.3. A region formation algorithm, like watershed or min-cut, is then used to produce a segmentation from the local boundary information, as described in section 3.4.

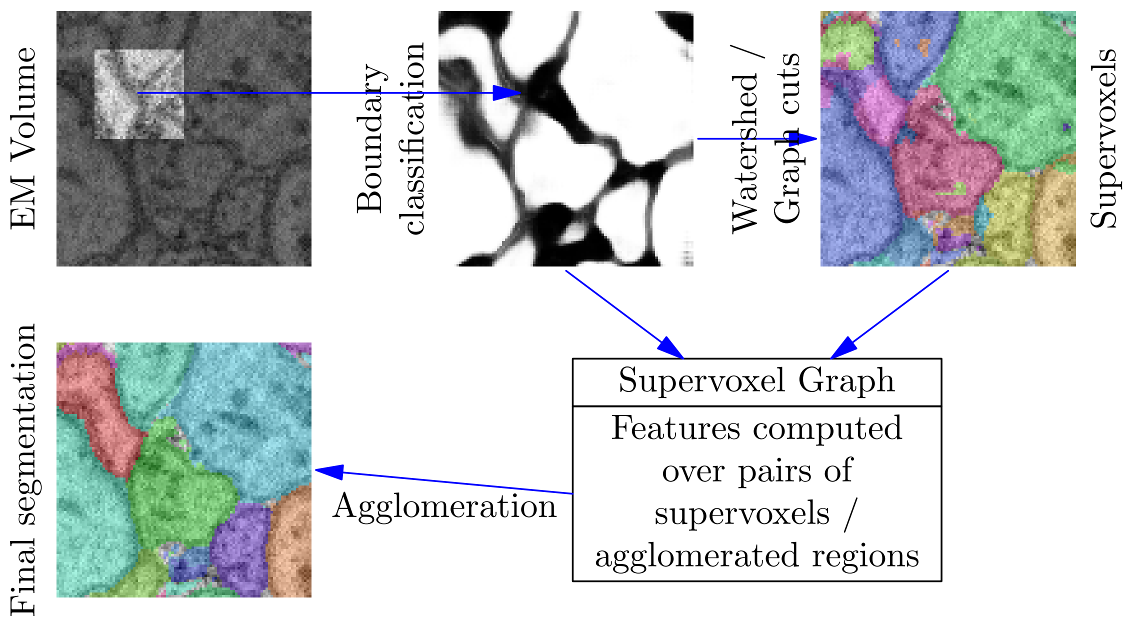
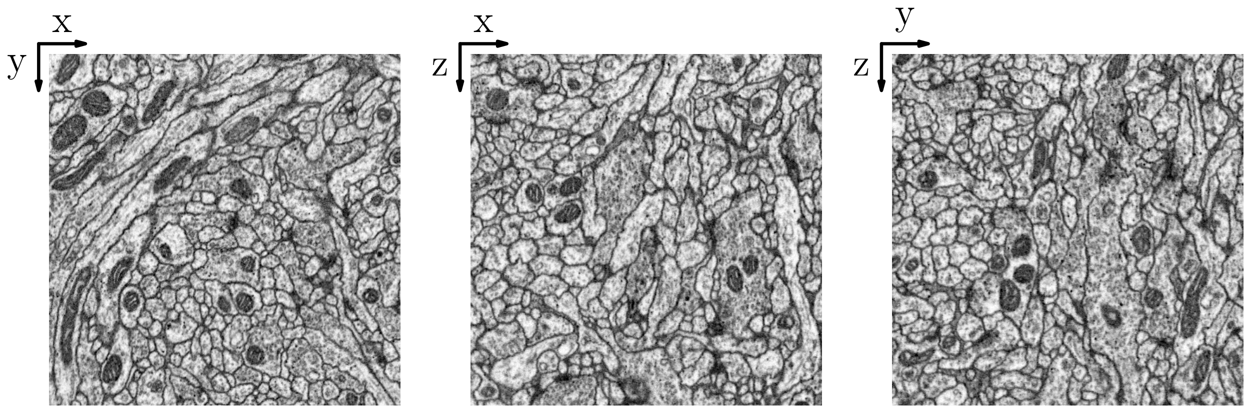


Figure 3.2: Architecture of prior agglomeration methods. As an extension of the boundary classification pipeline shown in fig. 3.1, the initial segmentation produced by the region formation algorithm is biased towards *undersegmentation* in order to produce 3D supervoxels or 2D superpixels. These serve as input to a region agglomeration procedure that merges adjacent regions based on image and object features. The region agglomeration approaches differ in how they perform agglomeration, and whether they compute static features over pairs of original supervoxels, or whether they compute features over agglomerated regions that change as these agglomerated regions grow, as described in section 3.5.

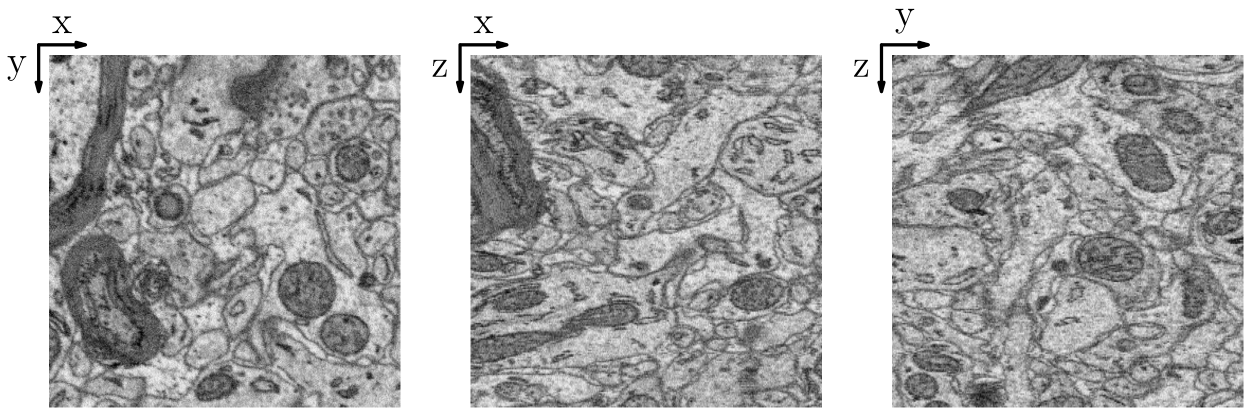
and there is a need to generalize over wide variation in appearance, but the expectations on accuracy are correspondingly much lower. Indeed, in comparisons against the specialized methods designed for electron microscopy, described in this chapter, state-of-the-art methods for natural scenes have not proven competitive, and in fact in some cases the methods designed for electron microscopy have achieved state-of-the-art performance even on natural scene computer vision benchmarks.

3.1 Rotational invariance and covariance

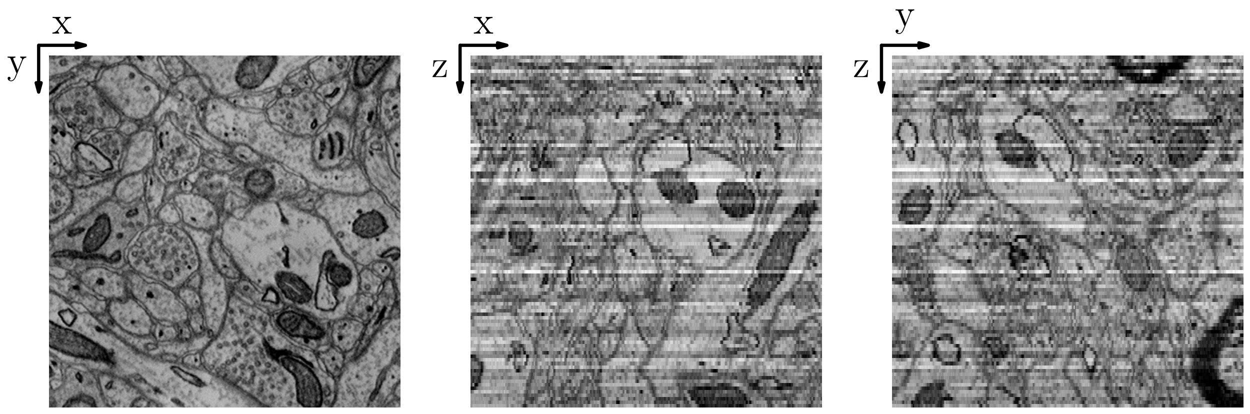
Many regions of the brain containing circuits of interest, such as the Mammalian cortex, have essentially no preferred orientation. More generally, as the scale is reduced, the extent to which there may be a preferred orientation is also reduced. There are notable exceptions, such as regions of the retina containing bundles of parallel axons, but even in such bundles, within the cross-sectional plane there may be no preferred orientation. Ideally,



(a) Drosophila larva, 10 × 10 × 10 nm FIB-SEM [68]



(b) Songbird, 15 × 15 × 25 nm SBEM (J0126 dataset)



(c) Mouse S1 cortex, 3 × 3 × 30 nm ATUM-SEM [55]

Figure 3.3: Cross-sectional views comparing rotational covariance of data obtained from different microscopy techniques. FIB-SEM (fig. 3.3a) is nearly perfectly covariant with axis-aligned rotations. SBEM (fig. 3.3b) and ATUM-SEM (fig. 3.3c) display strong covariance within the imaging plane, and covariance to reflection along the section axis z. At down-sampled resolution, SBEM is additionally somewhat covariant with respect to arbitrary rotations.

therefore, the results of automated reconstruction would be *covariant* with arbitrary orthogonal transformations of the data, meaning that transforming the input data by an arbitrary translation, rotation and/or reflection should have the effect of transforming the output in the same way. This corresponds to an assumption that the joint distribution over image data and corresponding reconstructions is *invariant* to arbitrary orthogonal transformations. The distinction between covariance and invariance is important: properties such as the total number of neurons or synapses are invariant to orthogonal transformations, while the morphology of all the neurons is covariant but not invariant to these transformations.

In practice, invariance and covariance properties of the data are limited by the rotational covariance of the imaging method, meaning the extent to which rotating the tissue prior to imaging yields the same result as digitally rotating the data after imaging.[1] Figure 3.3 shows cross-sectional views comparing the covariance properties of FIB-SEM, SBEM, and ATUM-SEM imaging.

FIB-SEM, which allows volumes to be imaged at isotropic resolution with pixels as small as 2 nm on a side, has nearly perfect covariance. For isotropic FIB-SEM data, it is therefore reasonable to assume covariance over the full octahedral symmetry group O_h, of order 48. Elements of this group correspond to an arbitrary permutation of the x, y, and z axes combined with arbitrary reflections of each of the x, y, and z axes. We can derive the order of the group as

$$|O_h| = \underbrace{3!}_{\substack{\text{permutations}\\ \text{of x, y, z}}} \cdot \underbrace{2^3}_{\substack{\text{reflections}\\ \text{of x, y, z}}} = 48.$$

Due to the anisotropic resolution of ssTEM (with typical resolution of $3 \times 3 \times 50$ nm) and ATUM-SEM (with typical resolution of $3 \times 3 \times 30$ nm), for those imaging methods we cannot assume covariance over the full ocahedral symmetry group, but we can reasonably assume covariance over 2-D rotations and reflections in x-y, and reflections in z; this corresponds to the D_{4h} symmetry group of order 16:

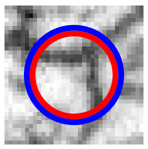
$$|D_{4h}| = \underbrace{2!}_{\substack{\text{permutations}\\ \text{of x and y}}} \cdot \underbrace{2^3}_{\substack{\text{reflections}\\ \text{of x, y, z}}} = 16.$$

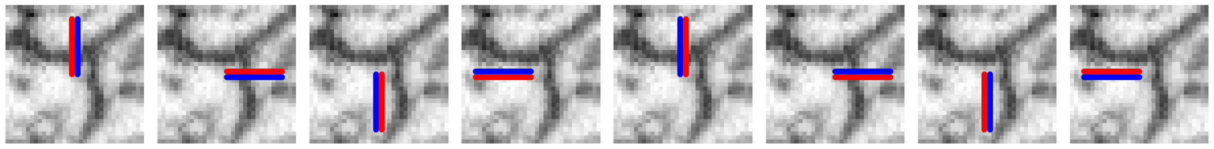
Because labeled training data is expensive to obtain (requiring tedious work by human annotators), it is advantageous for machine learning methods to take advantage of these covariance properties. There are three main approaches for doing this: rotationally-invariant features, explicit covariance, and data augmentation, as illustrated in fig. 3.4.

Rotationally-invariant features

Under the approach of using rotationally invariant features, the responses for each individual image feature are fully invariant to some set of rotations and reflections of the input image

[1] To avoid interpolation issues, we will consider only translations by whole-voxel amounts and *axis-aligned* rotations and reflections.


(a) Rotationally-invariant features


(b) Explicit covariance

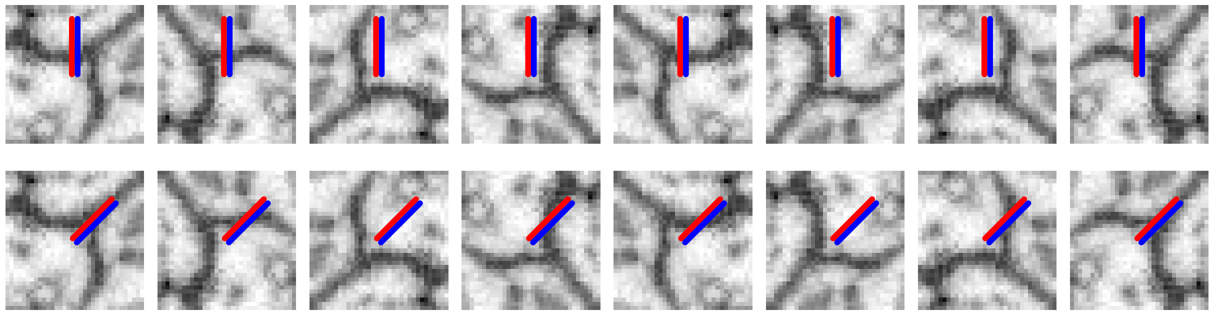

(c) Data augmentation

Figure 3.4: Approaches for handling rotational covariance and invariance. We compare a single rotationally invariant image feature (fig. 3.4a), a single rotationally covariant image feature (fig. 3.4b), and a single image feature under training data augmentation (fig. 3.4c). For illustration purposes we show only 2-D transformations, but in actual implementations 3-D transformations may be used as well. In the case of rotational invariance, each feature results in a single feature response that is invariant to rotations and reflections of the input about the center point. In the case of rotational covariance, each feature results in a separate feature response for each permitted transformation, shown left to right; transforming the input is equivalent to permuting the order of the feature responses. In the case of data augmentation, each feature results in a single feature response, as shown in the first row. To detect similar image features at a different orientation requires a completely independent feature, with no rotation-related parameter sharing, as shown in the second row. At training time, all features are trained and tested on each transformed version of the input, corresponding to the different columns. In practice this tends to result in there being, for each learned feature, approximately corresponding transformed versions. At test time, no data augmentation is performed.

about the center of the context region. When sampled at the center of the context region, the image intensity, gradient magnitude, structure tensor eigenvalues, and Hessian eigenvalues are examples of invariant features.

Forcing invariance in the image feature responses is highly advantageous in terms of computational efficiency and implementation simplicity: subsequent pipeline steps that depend only on the image features (but not the raw image) are automatically invariant to transformations of the image without any added complexity, and invariant representations tend to be relatively low dimensional. Forcing invariance at the level of the image features themselves does, however, severely limit representational power: for example, it is impossible to determine the direction of a boundary from rotationally-invariant image features, and therefore low-level invariant features are not suitable for computing higher-level information related to boundary continuity or shape.

Explicit covariance

Under the explicit covariance approach, covariance with respect to some group H of rotations and reflections is explicitly represented in the model, which we will describe very generally as consisting of a set of variables V, and a set of relationships R defined over ordered subsets of the variables. Variables correspond to inputs $W \subset V$ (including the raw image intensity values), any intermediate values, and outputs $Y \subset V$ (for example, a representation of a segmentation of the image). For $v \in V$, we will denote by $v(I)$ the realization of v for the input I. For each relationship $r \in R$, there is an ordered set $P(r)$ (possibly empty) of parameters. To specify the key covariance property, we will rely on the group-theoretic concept of group actions:

Definition 3.1 (Left group action [35, p. 41])**.** A *left group action* of a group G on a set A is a map $\varphi : G \times A \to A$ (written as $g \cdot a$ for all $g \in G$ and $a \in A$) satisfying the following properties:

(1) $g_1 \cdot (g_2 \cdot a) = (g_1 g_2) \cdot a$ for all $g_1, g_2 \in G$, $a \in A$; and

(1) $e \cdot a = a$ for all $a \in A$, where e denotes the identity element of G.

We denote by $X \subset \mathbb{Z}^3$ the coordinate space of the input and output. We denote by $T = \{t_x \,|\, x \in \mathbb{Z}^3\}$ the group of translations by integer offsets. There is a natural group action of the product group HT on X, where $ht_{x'} \cdot x$ is defined as the rotation/reflection h of $x + x'$ about the origin $\vec{0} \in \mathbb{Z}^3$, and a natural group action of HT on the input I. Regardless of how rotational covariance is handled, we make the assumption of translation covariance, which can be formally stated as there being a group action of T on V, satisfying

$$v(t \cdot I) = t \cdot v(I) \qquad\qquad \text{for all } t \in T,\ v \in V.$$

Explicit covariance in the model corresponds to the following properties:

1. there is a group action of the full product group HT on the set V of variables, satisfying
$$v(g \cdot I) = g \cdot v(I) \qquad \text{for all } g \in HT,\, v \in V;$$
2. there is a group action of HT on the set R of relationships satisfying
$$g \cdot \langle v_1, v_2, \ldots \rangle = \langle g \cdot v_1, g \cdot v_2, \ldots \rangle \qquad \text{for } \langle v_1, v_2, \ldots \rangle = r \in R.$$
3. the set of parameters $P(r)$ associated with each relationship $r \in R$ is invariant to the group action:
$$P(r) = P(g \cdot r) \qquad \text{for all } g \in HT,\, r \in R.$$

The special case of requiring rotationally-*invariant* features corresponds to the property that rotations of the input only *spatially* transform intermediate and output variables:
$$\forall g \in HT \colon \forall v \in V - W \colon \exists x \in \mathbb{Z}^3 \colon v(g \cdot I) = t_x \cdot v(I).$$

Explicitly encoding rotational covariance in the model effectively reduces the number of parameters to be learned by a factor of $|H|$, which may be as high as 48 in 3-D. This tends to significantly reduce training time. In contrast to rotationally-invariant features, there is no loss of representational power, though there is considerable added implementation complexity. Because covariance structure must be explicitly specified, however, there is some possibility of redundancy in the model, due to learned parts of the learned model that are coincidentally rotationally invariant (but not specified as such).

Data augmentation

Under the data augmentation approach, the model does not include any explicit handling of rotation, but is trained on an augmented training set that includes all permitted transformations of the original data. This approach is most general (in particular, it can equally well handle any type of transformation, including transformations that are not axis-aligned or non-rigid), simplest to implement, and unsurprisingly has been most widely used by prior work.

Compared to explicit covariance built into the model, however, data augmentation does increase computational and memory requirements during training by a factor roughly equal to the order of the transformation group, which can be as high as 48 in 3-D. This can make training of large models quite difficult in practice.

3.2 Slice-to-slice contour tracking

One of the earliest approaches considers the object reconstruction problem as a slice-to-slice contour tracking problem [52], closely mirroring the basic workflow of a human tracer.

Under this approach, it is assumed that all objects to be reconstructed are tube-shaped and that the volume is oriented such they all run approximately parallel to a particular axis (typically the depth/sectioning axis); this assumption is mostly reasonable for image volumes corresponding to bundles of parallel axons with cross sections perpendicular to the sectioning axis, for which the approach was designed.

Following the active contour model paradigm [53], an energy function is defined that assigns costs to 2-D closed contours. The energy is a combination of an internal energy term, equal to the weighted sum of the squared magnitudes of the first and second derivatives of the contour position as a function of arc length, which encourages smoothness, and an image-dependent term, based on edge and intensity information in the image. The energy is invariant to rotations and reflections in 2-D.

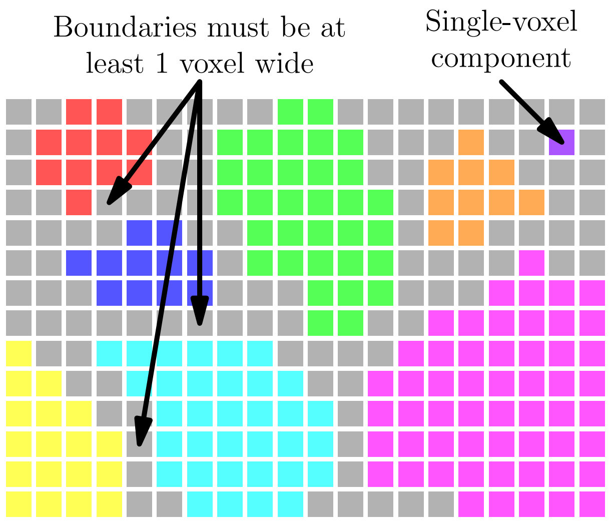
Starting from a user-specified seed point in the interior of an object in one slice, the algorithm repeatedly samples a set of contour points along rays originating at the seed point; these contour points define a simple closed curve. The sampled contour points are restricted to lie within a fixed distance range from the seed point, based on the assumed diameter of the object. The sampled contour that minimizes the energy function is chosen as the initial contour.

A procedure based on Kalman filtering [12] is used to propagate the contour from one slice to the next. The state and measurement vectors consist of 2-D positions and "velocities" for each contour point. The "measurement" for each subsequent slice is initialized by applying the 2-D velocities determined by optical flow [63, 19] to the contour state positions from the previous slice, and then refined by sampling to minimize the contour energy. The Kalman filtering makes the contour tracking more robust to poor contour fitting in a single slice due to e.g. poor image contrast.

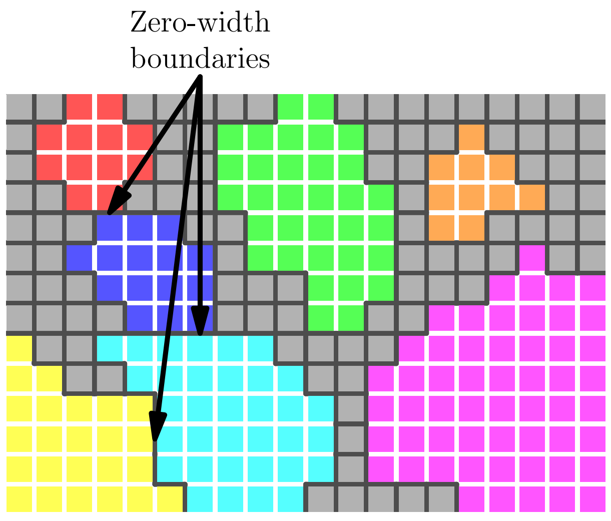
A key strength of this approach is that it relies on only a very small number of parameters that can be tuned by hand without the need for any training data. Its usability is severely limited, however, by the need to manually specify seed points, the inability to handle objects that branch, terminate, or do not run parallel to the sectioning, and the fact that all objects are tracked independently without any mutual exclusion.

3.3 Boundary classification

Given that cell membranes, which are specifically targeted by the heavy-metal stains used to provide contrast in electron microscopy, are a very direct and primary indication of neuron shape, it is natural to treat detection of boundaries as at least a key sub-problem. A wide variety of machine learning-based methods have been employed for this purpose. These methods all classify whether there is a boundary at some individual center position based on some fixed-size surrounding image context. To predict boundaries at all positions within the volume, the classifier is applied in a sliding window fashion. Depending on the structure of the classification model, in particular whether it involves some form of weight sharing such as convolutional filtering, it is in some cases possible to densely predict boundaries at



(a) On-voxel representation



(b) Between-voxel representation

Figure 3.5: Comparison of on-voxel and between-voxel boundary representations. Colors indicate the connected components implicitly defined by the boundary representation. In the on-voxel representation (fig. 3.5a), boundary/background voxels are shown in gray. In the between-voxel representation (fig. 3.5b), the presence of a boundary between neighboring voxels is indicated by a dark gray line, while the lack of a boundary is indicated by a white line. Voxels not connected to any neighbors are indicated in gray. With the on-voxel representation, boundaries must be at least one voxel thick, while the between-voxel representation allows zero-width boundaries. The between-voxel representation cannot distinguish between background voxels and single-voxel foreground components, and therefore such cases are always treated as background voxels.

all positions within an entire volume at a computational cost significantly lower than the number of boundary positions times the cost to predict at a single position.

Boundary representation: voxels vs. adjacent voxel pairs

One important representational choice that has been identified in prior work [82, 81] is whether boundaries are *on* individual voxels, i.e. a boundary takes up a voxel, or are *between* neighboring voxels, as shown in fig. 3.5. The former case, placing boundaries *on* voxels, can be seen as labeling voxels as either corresponding to foreground or background, with separate foreground objects necessarily separated by at least one background voxel. In the case of neuron segmentation, foreground voxels can be seen as corresponding to intracellular space, with background voxels corresponding to extracellular space.

In the latter case, placing boundaries *between* voxels, boundaries correspond to edges in a graph over voxels. Foreground/intraceullar space and background/extracellular space are not explicitly represented; instead, voxels not connected to any neighbors are assumed to be

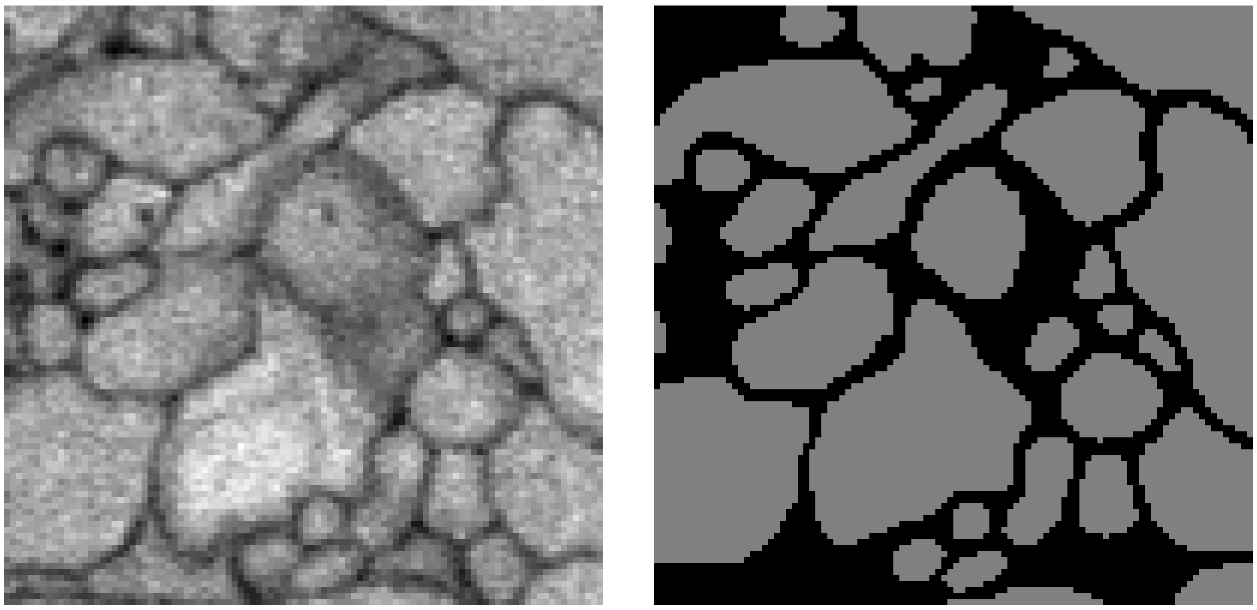

(a) Raw image (b) On-voxel boundaries

Figure 3.6: On-voxel boundary labeling as denoising. The image on the left is a 100^2 voxel region of a single section from the rabbit retina dataset E1088 [43], imaged using Serial Block Face Scanning Electron Microscopy (SBEM) at a resolution of $22 \times 22 \times 30$ nm. The tissue was prepared using a horseradish peroxidase (HRP) solution [18] to enhance cell outlines.

background space. As a consequence, it is impossible to represent single-voxel foreground components using the between-voxel boundary representation. For neuron segmentation from electron microscopy, this is not an issue in practice because a single-voxel neuron is not physically possible.

As has been noted in prior work [50], the on-voxel representation can be seen as a *denoised* version of the original image. This relationship is particularly clear in the case of volumes stained to enhance cellular outlines and hide intracellular structures, as shown in fig. 3.6. For objects near the limit of the image resolution, i.e. only several voxels wide, the ability of the between-voxel representation to specify zero-width boundaries can be important.

Prediction of on-voxel boundaries requires just a single classifier that predicts whether there is a boundary at the center voxel of some context region. In contrast, prediction of between-voxel boundaries requires, in principle, a separate classifier for each boundary direction. For example, in the case of 6-connectivity, in which boundaries are considered between adjacent voxels in the x, y, and z directions, but not diagonally, a total of 3 classifiers are needed to predict given some context region whether the center voxel is connected to the neighboring voxel in the positive x, y, and z directions. In practice, these classifiers may, however, share much of their computation.

It has been observed that using the between-voxel representation for machine-learning-based boundary classification yields better segmentation accuracy compared to using the on-voxel representation. [82, 81] One possible explanation is that the between-voxel representation may simply be more capable of representing the actual boundary structure, due to zero-width boundary issues. Another possible explanation is that the single on-voxel boundary classifier must learn to recognize boundaries in any direction. Because image cues relevant to detecting a boundary in one direction may not be very relevant for detecting a boundary in a completely different direction, the task may amount to learning a *mixture* of classifiers for several different directions. In contrast, for between-voxel boundary prediction, the separate per-direction models are explicit, which effectively amounts to a form of additional supervision during training.

Machine learning architectures

The training of classifiers to predict either on-voxel or between-voxel boundaries depends on image data with associated boundary information labeled manually by human tracers either densely or sparsely.

Random forest classification using hand-designed features

One approach to boundary classification that has been taken is to train a random forest classifier [15] on some vector of hand-designed image features computed over the context patch. Several different combinations of features have been used:

- One method [5] combines the original center intensity value, difference of smoothed intensity at center under two different Gaussian filters, gradient magnitude, sorted eigenvalues of the 3-D structure tensor computed over several different 3-D neighborhood sizes, sorted eigenvalues of the Hessian, and statistics of the distribution of intensity values and gradient magnitude within several different 3-D neighborhood sizes. This method also served as the basis for local image classification in the open-source tool ilastik. [78]
- Another method [83] uses an Adapted Zernlike feature vector obtained by keeping only a fixed number of low-frequency bands of the Discrete Zernlike Transform (DZT) [67] computed over a 2-D circular disk, and normalizing for rotation (but not reflection) based on the most prominent orientation within the disk.

Both methods rely on image features that are fully rotation invariant (though in the case of the DZT-derived features, only in 2-D), and consequently are used only to predict on-voxel boundaries. Because the feature vector dimensionality is kept low, these methods achieve reasonable results with only a small amount of training data; because the model capacity is correspondingly limited, however, the improvement in accuracy from larger amounts of training data is also quite limited.

In addition to detecting boundaries, these same classification architectures have also been employed for other types of classification in electron microscopy images, such as detection of mitochondria and glial cells.

Neural networks

Convolutional neural networks have been widely used for boundary classification, using either on-voxel [50, 48, 23] or between-voxel boundary representations. [82, 81] There has been significant focus on the loss function used for training: in addition to a simple independent loss per boundary classification (i.e. pixel error) [50, 23], loss functions based on differentiable relaxations of the Rand Index [82] and warping error [48], as well as a weighting based on "local error density" [47] have also been employed.

Sequential networks in which successive networks are trained to correct the output of the previous network have also been employed with some success. [76, 47] In addition to networks operating directly on the raw image intensity values, other derived features have also been employed, including sparse coding representations of patches [47] and a rotation-invariant Radon-like feature representation [61, 76].

When a large amount of training data is available, neural networks have generally yielded the best accuracy for boundary classification, though the performance is highly dependent on the specific network architecture and the time required to train the networks is considerable (on the order of weeks). While building explicit covariance into these models is possible in principle, it has not yet been done; prior work involving neural networks has so far relied only on the simpler approach to covariance of data augmentation.

3.4 Region formation

Boundary predictions are only one part of a pipeline for reconstructing neural circuits. To convert the real-valued boundary prediction scores to a discrete segmentation, a region formation procedure is required. The simplest such procedure is to compute connected components based on thresholding of the boundary predictions. [50, 82, 48] One argument in favor of an extremely simple region formation procedure is that it better matches the loss function used to train the boundary classifier; it is difficult to directly account for more sophisticated procedures in the training objective. A critical disadvantage, however, to pure connected components is that it tends to result in many tiny, undesired components in certain areas, and at the same time is also susceptible to falsely merging two components based on a single boundary prediction false negative.

Variants of the watershed transform [11, 10] have been employed [83, 4] as a somewhat more sophisticated alternative to connected components. The main benefit is the consolidation of tiny components, though marker-based watershed, using heuristics to place the marker points, does serve to limit undersegmentation (at the cost of introducing much greater undersegmentation).

Another approach [64, 37] uses a min-cut/max-flow algorithm [14] to partition 2-D slices into intracellular and extracellular space, i.e. on-voxel boundary labeling. The connected components of the intracellular space define the segments.

3.5 Agglomeration

While the segmentations produced by the region formation algorithms have in some cases been used directly as the final output, this is obviously insufficient in the case of 2-D region formation, and even for 3-D region formation the accuracy obtained using local boundary information alone tends to be rather poor. To improve accuracy, a variety of agglomeration methods have been proposed. The initial region formation is biased towards *undersegmentation* in order to produce 3-D supervoxels or 2-D superpixels. These serve as input to a region agglomeration procedure that merges adjacent regions based on image and object features.

There are several key advantages to operating on supervoxels, rather than individual voxels:

- poor local optima are less likely to be a problem;
- features can readily be computed over larger areas;
- computational efficiency due to the smaller number of decisions that must be considered.

Three main approaches to agglomeration have been used by prior work: independently predicting whether to merge each pair of adjacent supervoxels, globally optimizing the set of supervoxel merges to perform based on certain constraints, and applying a learned policy to incrementally merge adjacent regions.

Independent agglomeration

The simplest approach to agglomeration is to treat whether to merge each pair of adjacent supervoxels as independent decisions to be predicted by a trained binary classifier.

One prior method [5] relies on a random forest classifier using a collection of hand-designed features computed over *pairs* of adjacent supervoxels:

- values derived from the number of voxels in each supervoxel and the length of the boundary between them;
- statistics of the raw image intensity within each of the two supervoxels; and
- statistics along the boundary between the two supervoxels of the raw image intensity values, gradient magnitude, largest eigenvalue of Hessian, and boundary prediction scores.

Another prior method [64] (intended for the simpler task of segmenting individual mitochondria rather than whole neurons) relies on a Support Vector Machine (SVM) classifier using a different collection of features, computed *independently* for each supervoxel:

- a vector of ray descriptors [77] computed based on supervoxel shape and boundary predictions, normalized for rotation (but not reflection);

- the vector of responses from oriented edge filters at the supervoxel center, normalized for rotation; and
- statistics of the raw image intensity within the supervoxels.

The SVM classifies whether to merge two supervoxels based on the combination of both of their feature vectors.

Because all merge decisions are made independently, training and testing are quite straightforward for these two methods, but the improvement in accuracy is also limited.

Global optimization

One approach to improve the accuracy of independent agglomeration is to optimize a global objective over all merge decisions subject to certain consistency constraints. A trained or heuristically-defined classifier is still used to score pairs of adjacent supervoxels, but instead of using these scores directly to determine which merges to include, as in independent agglomeration, the score is treated as an additive per-pair cost/reward that is added over all merge decisions to define the global objective.

One prior method [4] formulates this optimization as a correlation clustering problem [7], which enforces transitive consistency of merge decisions: if a supervoxel a is connected to a supervoxel b to which it is also adjacent, then a and b must be merged. While correlation clustering is NP-complete, an approach based on iteratively solving an integer linear program (ILP) to which violated constraints are successively added proved highly effective in practice. Transitive consistency is a very powerful constraint for limiting undersegmentation, particularly in the case of large objects between which there are many pairs of adjacent supervoxels. It does, however, have the disadvantage of being potentially *too* restrictive, in that a neuron touching back on itself, as shown in fig. 3.7, violates this constraint.

Another prior method, called segmentation fusion [83], uses a different formulation designed specifically for anisotropic-resolution volumes (such as may be obtained using ssTEM or ATUM-SEM):

- sets of *possibly overlapping* 2-D superpixels, at multiple scales, are found for each 2-D section, by computing watershed segmentations at multiple thresholds based on boundary classification scores (in highly anisotropic volumes, performing 3-D segmentation directly based on boundary scores may not yield reasonable results);
- an ILP is used to select
    1. a *non-overlapping subset* of the superpixels in each section; and
    2. a *consistent* set of edges linking superpixels that overlap between adjacent sections.

Within each 2-D section, only a non-overlapping subset of the candidate superpixels for that section may be chosen. Edges link superpixels between adjacent 2-D sections, but not within

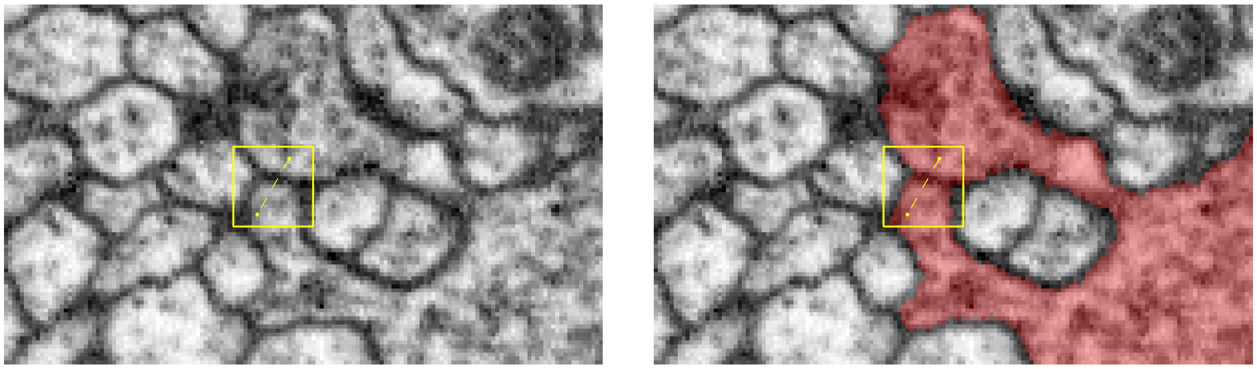


Figure 3.7: Distinction between local and global connectivity. In the cross-section of raw data on the left, there is clear evidence that the two points indicated within the yellow bounding box are separated by cell membrane. From the manual annotation overlaid on the right, it is clear, however, that they are nonetheless part of the same cell, highlighted in red. Thus, within a sufficiently local area the two points are disconnected, but globally they are connected. Distinguishing the connectivity of points at *multiple scales* is critical for accurate shape modeling. If connectivity is represented only globally, as in prior agglomeration work [4, 68], it may be impossible to reconcile strong local evidence of a cell boundary between two parts of the same sell in cases of self-contact, leading to poor learning and incorrect predictions for these cases.

sections; larger segments within each 2-D section are possible only to the extent that there are larger superpixels in the candidate set. The optimization objective is to maximize the sum of heuristically-defined non-negative rewards corresponding to each selected superpixel and each selected edge. The key consistency constraint is that each superpixel is linked to at most one superpixel above and at most one superpixel below, with the result that the resultant 3-D components are effectively limited to being tubes. A superpixel linked in only one direction corresponds to an end of such a tube.

This constraint strongly limits the propagation of false merge errors, and is readily supported by standard ILP solvers without the need for any special constraint sampling. There is the key limitation, however, that branching structures of neurons are simply not handled at all by this approach.

A variant of this approach, called SOPNET [37], attempts to handle branching by relaxing the consistency constraint to allow linking a superpixel in one section to at most 2 different superpixels in the section above, and at most two different superpixels in the section below. Rewards, computed as a function of various hand-designed features of the segments, are learned for:

- regular one-to-one links;
- split/merge links between two superpixels in one section and a single superpixel in an

adjacent section.

- terminations, corresponding to a superpixel without any link in a given direction.

Learned-policy agglomeration

An alternative to the optimization view is to consider agglomeration as a sequential decision problem for which a policy must be learned. [49, 68, 13] Starting from an initial overseg-mentation (consisting of 2-D superpixels or 3-D supervoxels), a learned policy sequentially chooses to merge a pair of nearby regions (or to stop agglomeration); the policy simply pre-dicts a score for each pair of nearby regions based on some set of features, and chooses the best scoring pair, or terminates if the best score does not exceed some threshold. Notably, under this approach there is no global objective defined over segmentations that is being optimized.

A key strength of this approach is that features (and consequently the pairwise merge scores) are recomputed with respect to the larger agglomerated regions as agglomeration proceeds; scores computed with respect to larger regions are potentially more reliable. In contrast, the methods based on integer linear programming rely on fixed scores defined based on the original superpixels/supervoxels, just like independent pairwise agglomeration.

Features computed over pairs of regions by these methods include:

- the number of voxels in each of the two regions; [49, 68, 13]
- distance between the centroids or closest points of the two regions; [49]
- statistics of the boundary classification scores (and other classification scores, like pres-ence of mitochondria) within each region or along the boundary between the two re-gions; [49, 68, 13]
- measures of the primary orientation of each region; [49, 68, 13]
- shape representations of the two segments based on ray descriptors or log-polar binned histograms of boundary locations; [13]
- the raw image data, the mask for each of the two regions, and/or local image/joint image and region mask features pooled over the segments or along their boundaries. [13]

Chapter 4

Error Metrics for Segmentation

Error metrics for evaluating the accuracy of segmentations are critical for:

- determining whether the segmentation is of sufficient accuracy for use in deriving scientific results;
- comparing the effectiveness of different methods or variants (i.e. for *human* learning);
- defining optimization objectives for machine learning.

Prior work has investigated the accuracy of humans at segmenting neurons in electron microscopy volumes. [43, 41, 44] Despite the frequent occurrence of both attention-related and difficulty-related errors by human annotators, particularly non-expert ones, consensus results obtained from multiple annotators appear to be highly reliable. While it is impossible to rule out correlated errors between multiple human annotators, existing automated methods are in any case significantly less accurate, and therefore it is reasonable to evaluate accuracy in terms of an assumed *ground truth* segmentation obtained by manual annotation. These ground truth segmentations may be in the form of *dense reconstructions* that precisely mark all voxels belonging to each neuron, or *skeleton reconstructions* that specify just the approximate center line of each neuronal process by a set of 3-D line segments.

Even with a ground truth segmentation, determining a reasonable error metric for evaluating an automated segmentation is a non-trivial problem. Many scientific questions depend on the reconstructed circuit graph specifying the synaptic connections between neurons. The accuracy of this connectivity information is directly related to the chance that there is a path from a synapse to the correct soma (cell body), and no additional paths to other somas. In principle such a metric could be computed based on ground truth skeletons or dense segmentations, but because the distance from a synapse to the soma may be extremely large, this metric has several disadvantages of in practice:

- fully tracing many such synapse to soma paths manually would be very expensive;
- it is not suitable for evaluating accuracy on small volumes;

- it is meaningful only if the overall accuracy is sufficiently high that a reasonable fraction of synapse to soma paths are correct.

The latter two disadvantages in particular make this metric unsuitable for defining a training objective function, as a not-fully-trained model can be expected to have low accuracy.

For evaluation using smaller volumes and for machine learning, prior work has therefore considered a number of alternative error metrics.

4.1 Per-voxel error

One very straightforward approach to evaluating segmentation accuracy is based on the boundary representation of a segmentation (section 3.3): predictions for the presence of a boundary at each voxel position (for an on-voxel representation) or between each pair of neighboring voxels (for a between-voxel representation) are treated as independent classification problems.

This approach has the advantage of being simple to implement and entirely local, such that it can reasonably be used with small volumes, or even just a sparse set of locations with boundary labels. It cannot, however, be used with ground truth specified by skeleton reconstructions, since skeletons do not actually specify boundary locations.

For actually validating the accuracy of a segmentation, per-voxel boundary accuracy is problematic, because it is highly sensitive to the precise boundary position, but relatively insensitive to holes in boundaries, which can result in false mergers. Specifically, shifting the position of a long boundary by a single voxel may result in a large change in boundary accuracy, while adding a single-voxel hole in a boundary will result in a very small change in boundary accuracy. Despite these shortcomings, it has been used quite successfully for defining a training objective for boundary classifiers, particularly when combined with appropriate balancing of the boundary and non-boundary class weights. [50, 5, 23]

4.2 Rand Index

Rand Index [70] is a similarity measure between two arbitrary clusterings (i.e. partitions) of a common set of points.

Definition 4.1. Given partitions X, Y of a set A, the Rand Index of X and Y is defined as

$$\frac{1}{\binom{|A|}{2}} \left|\left\{\{a,a'\} \in [A]^2 \,\middle|\, (a \equiv_X a' \wedge a \equiv_Y a') \vee (a \not\equiv_X a' \wedge a \not\equiv_Y a')\right\}\right|,$$

where $a \equiv_X a'$ denotes whether a and a' are contained in the same subset in X.

When applied to segmentations of 2-D or 3-D images, with the points A corresponding to voxels and the clusters corresponding to regions, the Rand Index is the fraction of pairs of voxels that are either connected (i.e. part of the same region) or disconnected (i.e. part of different regions) in *both* segmentations.

Because the Rand Index is constrained to the interval $[0, 1]$, a Rand error can be defined as 1 minus Rand Index, equal to

$$\frac{1}{\binom{|A|}{2}} \left| \left\{ \{a, a'\} \in [A]^2 \,\middle|\, (a \not\equiv_X a' \wedge a \equiv_Y a') \vee (a \equiv_X a' \wedge a \not\equiv_Y a') \right\} \right|,$$

where $\{\{a, a'\} \in [A]^2 \mid a \not\equiv_X a' \wedge a \equiv_Y a'\}$ is the set of falsely merged (connected) pairs in Y relative to a ground truth segmentation X, and $\{\{a, a'\} \in [A]^2 \mid a \equiv_X a' \wedge a \not\equiv_Y a'\}$ is the set of falsely split (disconnected) pairs in Y relative to X.

Compared to the per-voxel boundary mask error, Rand Index has the advantage of being much less sensitive to the precise boundary position, and much more sensitive to holes in boundaries, as shown in fig. 4.1. It also does not require a dense ground truth segmentation; it can be evaluated on ground truth skeleton reconstructions, although if the skeletons are sparse the extent to which the Rand Index metric detects false merges may be limited.

A key result shown in prior work [82] is that a subdifferentiable structured classification loss can be defined based on the Rand Index, and that neural network boundary classifiers can be trained to minimize this loss. Compared to a per-voxel boundary loss, this inherits the advantages of Rand Index, but has the disadvantage of making the objective much less stable, which in practice makes training significantly more difficult.

4.3 Variation of Information

Variation of Information [65] is an alternative information-theoretic distance metric between two clusterings.

Definition 4.2. Given partitions $X = \{X_1, X_2, \ldots\}$ and $Y = \{Y_1, Y_2, \ldots\}$ of a set A, the Variation of Information between X and Y is defined as

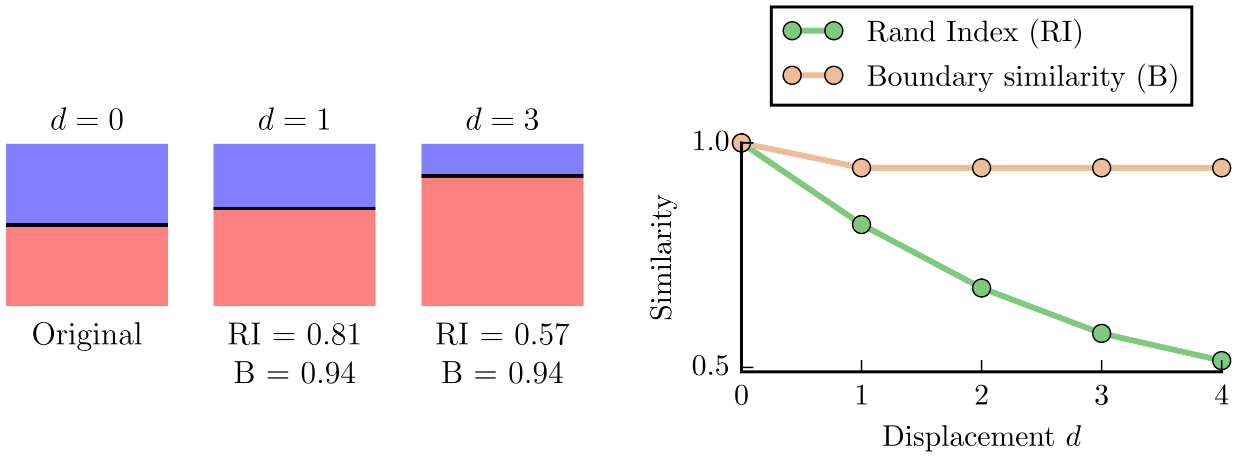
$$\mathrm{VI}(X, Y) \coloneqq -\sum_{X_i \in X} \sum_{Y_j \in Y} \frac{|X_i \cap Y_j|}{|A|} \left[\log_2 \left(\frac{|X_i \cap Y_j|}{|X_i|} \right) + \log_2 \left(\frac{|X_i \cap Y_j|}{|Y_j|} \right) \right].$$

Given a uniform distribution over $a \in A$, if we define random variables i and j corresponding to the events $a \in X_i$ and $a \in Y_i$, respectively, then

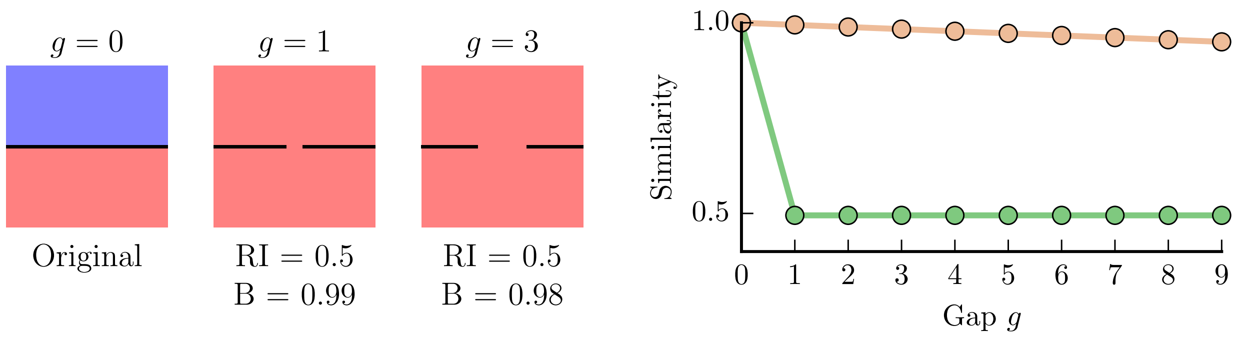
$$\mathrm{VI}(X, Y) = H(i|j) + H(j|i),$$

where

$$H(i|j) = -\sum_{X_i \in X} \sum_{Y_j \in Y} \frac{|X_i \cap Y_j|}{|A|} \log_2 \left(\frac{|X_i \cap Y_j|}{|Y_j|} \right).$$



(a) Sensitivity to boundary displacement. We compare the effect on Rand Index (RI) and boundary similarity (B) of displacing a boundary by d pixels (out of a total height of 10), without affecting the topology. The Rand Index decreases gradually as the displacement increases, while even a single pixel displacement causes the boundary similarity to jump to its minimum value.



(b) Sensitivity to boundary gap. We compare the effect on Rand Index and boundary similarity of introducing a g-pixel gap in 10-pixel wide boundary. Any non-zero gap drastically changes the topology and number of connected components. Consistent with this, the Rand index jumps immediately to the minimum value with any gap; in contrast, the boundary similarity decreases very gradually.

Figure 4.1: Comparison between Rand Index and per-voxel boundary similarity, equal to the fraction of boundary locations that match, on several simple 10×10 pixel examples using a between-voxel boundary representation. We compare the sensitivity of the two metrics to displacement of the boundary (fig. 4.1a) and introduction of a gap in the boundary (fig. 4.1b).

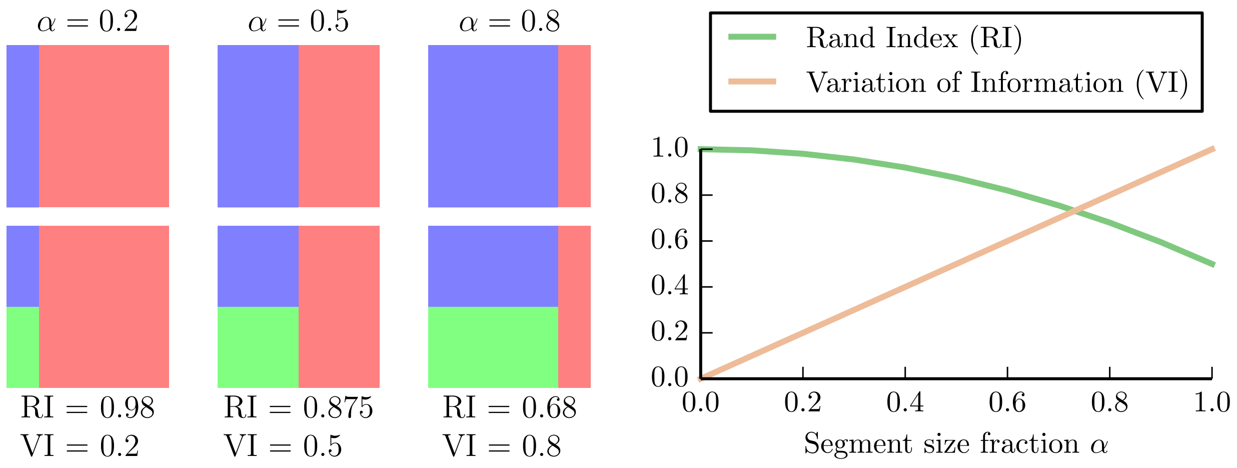


Figure 4.2: Comparison between Rand Index and Variation of Information of segment scaling behavior. In the top row the blue segment is an $\alpha \in \{0.2, 0.5, 0.8\}$ fraction of the total area; the red segment takes up the remaining $1 - \alpha$ fraction. In the bottom row, the blue segment is split into two equal parts, while the red segment remains unchanged. We show the Rand Index (RI) and Variation of Information (VI) between each segmentation in the top row and the corresponding segmentation in the bottom row. The Variation of Information $\text{VI} = \alpha$ is exactly equal to the fraction α of the area that was split, while the Rand Index $\text{RI} = 1 - \alpha^2/2$ decreases as a quadratic function of α.

If we consider X to be the ground truth segmentation and Y to be the predicted segmentation, then $H(X|Y)$ corresponds to the false mergers in Y of components of X, and $H(Y|X)$ corresponds to false splits in Y of components of X.

Both Rand Index and Variation of Information have been used by prior work for evaluating segmentations of neural tissue. [3, 68] A key advantage of Variation of Information over Rand Index, recognized by prior work [68], is that it effectively weights each segment linearly in its size, while Rand Index effectively weights each segment *quadratically* in its size, as shown in fig. 4.2. Because neuronal processes within the same volume can vary greatly in size, the quadratic weighting of Rand Index can be undesirable, particularly when evaluating large volumes.

Chapter 5

Scalable Wide Sparse Learning

In this chapter, we introduce a highly-scalable, layered architecture for classification on 3-D volumes: unlike conventional dense deep learning approaches, we rely on simple, parallelizable clustering algorithms and convex optimization to learn wide, sparse models. This architecture addresses a key limitation of prior work: lack of sufficiently large and expressive models. We avoid the largely unparallelizable bottleneck of stochastic gradient descent that has limited approaches to this problem based on dense convolutional neural networks to a small number of filters. We exploit rotational invariance of the data distribution by explicitly representing rotational covariance in the model, which reduces computational and memory requirements during training by 48 fold. This, in combination with a highly-optimized GPU implementation of certain sparse convolution-like operations, enables our approach to easily scale to models with 10000 or more (convolution-like) channels at each layer.

5.1 Methods

Our full architecture for the electron microscopy segmentation task is shown in Figure 5.1. Each node (corresponding to a box in the diagram) has a set of channels, and has a value for each channel and position within the data volume. Depending on the type of node, the position domain may be the set of voxel positions, the set of edges (using 6-connectivity) between two adjacent voxels, or the set of corners between voxels. Our approach consists of three main components: k-means-based quantization of all overlapping patches of the image data for use as classification features; supervised feature pooling based on sparse multiclass classification; classification of edges between voxels, using the k-means quantization features directly as well as the output of the pooling layer. The final segmentation is produced by connected components applied to the thresholded edge scores. The structure is similar to a convolutional network, but some of the “filters” are not linear, and due to the sparsity, an implementation of the linear parts using regular convolutions would be extremely inefficient.

We argue against the commonly state belief that training time is not an important consideration. Under the assumption that a model will be trained once and then applied to

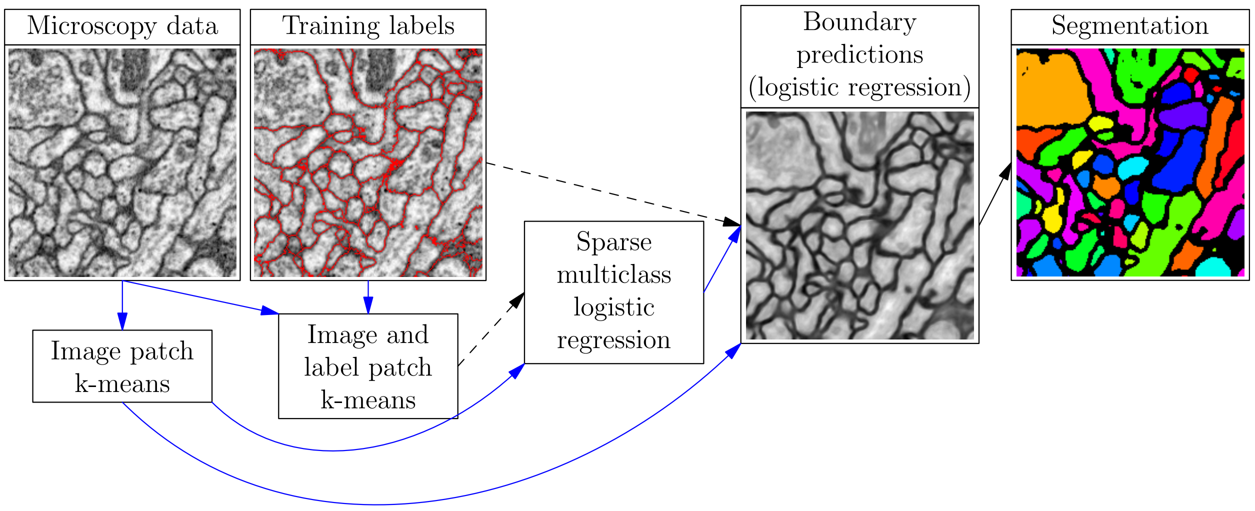

Microscopy data
Training labels
Boundary predictions (logistic regression)
Segmentation
Sparse multiclass logistic regression
Image patch k-means
Image and label patch k-means

Figure 5.1: Full architecture for the electron microscopy segmentation task represented as a computation graph. Blue lines indicate that each position in the output depends on a patch of the input centered at that position, while black lines indicate that each position in the output depends on only a single input position. Dashed lines indicate information used only for training the model. 2-D slices are shown as illustrations, but all nodes actually operate on 3-D patches. The final classifier actually predicts boundaries between adjacent voxels, and only an approximation can be shown.

a very large amount of test data, training time is indeed not very significant. However, this assumption rarely holds in practice: machine learning approaches generally have to be tuned carefully to each particular setting. In some cases, the choice of hyperparameters can even be more important than the training itself.[69] When the training of each particular variant takes days, weeks, or even months, this tuning becomes impractical. For the particular application of mapping neuroanatomy in electron microscopy volumes, the microscopy techniques are still in a state of flux, and it may also be desirable to quickly retrain a model when more labeled data is produced or existing labeled data is corrected.

For this reason, our architecture does not rely on conceptually complicated algorithms; rather, it is designed to use only components that can scale to large model sizes and can be effectively parallelized on a cluster of machines, at test time, *and also during training.*

k-means clustering of the image data

Recent results[26, 24, 25] have demonstrated the effectiveness of k-means for computing feature representations. On natural images, it has been found to be important to use it in conjunction with whitening and contrast normalization. On the neuroanatomy electron microscopy datasets, this does not appear to be the case: whitening methods only seem to amplify noise, possibly because the fine intracellular structures too small to be modeled explicitly are nonetheless poorly modeled as Gaussian noise. Lighting and the unreliability of shading information in natural images are also not relevant considerations for microscopy. Provided that microscopy conditions are carefully controlled, the absolute intensity at a voxel can be highly informative. In cases where some correction is needed, it is likely better to do it explicitly and globally as a preprocessing step rather than implicitly as part of feature extraction.

Based on initial experimentation and these intuitions, to compute our primary feature representation, we used k-means with the L_1 distance metric applied to all overlapping fixed-size ($8 \times 8 \times 8$ in our experiments) patches of the raw data, using very large numbers of clusters (up to 20000). This process is completely unsupervised. Although soft k-means is often reported to work somewhat better than hard k-means, we nonetheless used hard k-means, as having a very sparse representation is critical to the efficiency of our approach. It is *patches* of these quantization identifiers, which we refer to as *primary features*, that serve as input to subsequent layers of the network.

The bottleneck in both learning the clusters and computing the quantization of all patches once the clusters have been learned is the computation of the distance between each overlapping patch and each cluster. Unfortunately this cannot be computed efficiently for L_1 distance using the Fast Fourier Transform as it can for L_2 distance. Instead, direct computation is needed, a task that requires an extremely high number of arithmetic operations (for 20000 clusters and $8 \times 8 \times 8$ patches, over 10 million sum of absolute difference operations per voxel). It is also a task ideally suited to GPU computation due to its SIMD (single instruction multiple data) structure and the extremely high ratio of computation to memory access. As this is not precisely a standard linear algebra operation, and we are unaware of an existing optimized implementation [1], we used a code generation approach to produce our own implementation that could be adjusted for different blocking strategies in order to maximize performance for each patch size. Due to the data sharing from overlapping patches, by having each GPU core operate on a small 3-D block of data points and a block of (around 8) consecutive clusters at a time, and by operating on one row (the last dimension of the 3-d array that is contiguous in memory) at a time using an unrolled loop, we achieved an extremely high arithmetic to memory access density, such that performance was limited only by the peak GPU performance of the underlying element-wise operation, i.e. sum-of-absolute-differences in the case of L1 distance ($\approx$ 225 billion elements/s per GPU), on the NVIDIA Tesla M2050 GPUs on which we ran the experiments.

A tree-based or hashing-based approximate nearest neighbor method could in principle be used as an alternative to the brute-force computation we use, but given the relatively small number of clusters (up to 20000 in our experiments), and the near impossibility of fully exploiting GPU parallelism with such methods, we do not think it likely that such methods would improve performance.

[1]Vector quantization, and the underlying vector distance computation are such basic operations that there are no doubt countless public and private GPU implementations in some form or another, we are not aware of prior work specifically relating to 3-D overlapping patches, for which our implementation is optimized.

Supervised feature pooling

The primary features described in the previous section are inherently limited to providing information about a very local region of the data, and it is not practical to try to increase the size of this region increasing the patch size much beyond $8 \times 8 \times 8$, due to the curse of dimensionality. Furthermore, it is desirable to focus any representation of large regions of the data on precisely what is most relevant for distinguishing the different structures in the data relevant to our learning task. Therefore, we propose a simple *supervised* feature pooling approach in which we first jointly cluster patches of the training data *with their labels* using a combined distance metric on both the data and the labels, and then train a multiclass classifier to predict the joint class at each position from just the primary features.

For the microscopy data, we used a weighted combination of L_1 distance on the image patch and Hamming distance on the label patch (which specifies the connectivity of all pairs of adjacent voxels within the patch), with the relative weighting chosen so that the contribution to the average distance from each of the two parts is approximately equal at convergence. The motivation for jointly clustering both the image and label data, rather than the label data alone, is that the labels only specify neuron boundaries, and some structures may correspond to the same label pattern but have very different appearances. For instance, mitochondria are labeled as just intracellular space, but appear as very dark structures with a characteristic shape and appearance in the data.

To predict the joint class from the primary features, we train a multiclass logistic regression model, allowing each class to depend on only the n (up to 10000 in our experiments) most-correlated features, where each selected feature corresponds to given a image-only cluster and an offset from the center of the joint patch being predicted. Without this feature selection, the model would be unreasonably expensive to train and due to the extremely high number of parameters, likely to overfit. We use a sparse representation that explicitly lists which joint class/feature pairs are present at each position in the training data, which requires a large amount of memory but enables very efficient evaluation of the likelihood objective and gradient. To optimize the model, we use L-BFGS and distribute the data over a cluster of machines. Our pooled feature representation simply specifies the maximum probability class under the learned model at each position. Computing the maximum probability class at each position can be done very efficiently and does not require a large amount of memory.

Boundary classification

Patches of both the primary and pooling features serve as input to a final singleclass logistic regression classifier for predicting the segmentation boundaries of neurons. Because this is a singleclass classifier, we can afford to use dense weights at this stage. We optimized the logistic regression model using L-BFGS and used a combination of L_1-regularization, L_2-regularization, and early stopping for regularization. Computing classification scores and gradients are essentially sparse convolution operations. We are not aware of any publicly-available optimized implementation of these operations, so we implemented our own optimized GPU implementations. Sparse operations are inherently limited by memory bandwidth, which makes them unable to take full advantage of the processing power of a GPU, but a GPU implementation can still be advantageous due to the typically several times higher memory bandwidth of a GPU compared to a CPU.

For forward computation of the logistic regression classifications, we precompute a sparse array representation of the feature values such that each GPU core could efficiently operate on disjoint 3-d blocks of the output, eliminating the need for slow atomic operations. Furthermore, the precomputed sparse representation divides each block of feature values into independent sets such that only a single synchronization barrier between GPU threads is needed after each independent set, rather than after each element. Performance was primarily limited by GPU memory bandwidth in loading the model weights ($\approx$ 150GB/s per GPU).

For computation of the feature weight gradients from the classification score gradients, we precomputed a different sparse array representation of the feature values such that each GPU core could operate on a block of model weights (stored in registers). Performance was primarily limited by GPU memory bandwidth in loading the classification score gradients ($\approx$ 150GB/s per GPU).

Classification objective

As described in section 3.3, the choice of boundary representation is a key modeling decision. Prior work[81] that has concluded that classifying between-voxel boundaries rather than on-voxel boundaries yields better performance, and we have confirmed this to be the case as well empirically in initial experiments, and believe one reason is that it allows each of the three classifiers to specialize on a particular direction of boundary edge, and the "average" appearance of boundary edges at different orientations is not necessarily a likely boundary. We therefore chose to use a between-voxel representation that (in principle) requires training three separate classifiers for boundaries in the x, y, and z directions, though due to rotational invariance of the data, these may not be independent.

Another key modeling choice is the objective function for the classifier. As discussed in chapter 4, trying directly maximizing the per-boundary label likelihood is a simple choice, but has the key disadvantages of sensitivity to precise boundary location and insensitivity to gaps in boundaries. Prior work has suggested a relaxation of the Rand Index [82] and an approximation of a topology-based warping error [48] as better choices for defining an objective. However, neither metric allows for a convex objective, which eliminates any convergence guarantees, and we found in practice both metrics make learning much less stable and much more sensitive to optimization parameters.

Both of these error metrics can be seen as *dynamically* reweighting the boundary locations. We take the simpler approach of defining a *static* reweighting of boundary locations, based on similar intuition to warping error. Specifically, the true boundary edges between an intracellular voxel and an extracellular voxel or a voxel in a different segment are given the weight w_b. Edges between two extracellular voxels are given 0 weight. To compute the weights for intracellular edges, we dilate the boundaries by up to 2 voxels under the constraint that the boundary topology does not change. Edges between two intracellular voxels where at least one of them is not within the dilated region are given weight w_i. Remaining intracellular edges are given 0 weight. The weights w_i and w_b are chosen such that the total weight for boundary and non-boundary edges is equal. This simpler approach has the key advantage of retaining convexity of the objective.

Rotational invariance/covariance

As described in section 3.1, taking advantage of rotational invariance and covariance properties of the data has the potential to significantly reduce the amount of training data required as well as computational costs. We took the approach of enforcing directly that the model is fully rotationally *covariant*, without enforcing rotational *invariance* of the features.

We explicitly specify the orbit structure of each channel set C (corresponding to the set of k-means clusters). Each orbit is isomorphic to a subgroup of G. Applying a transform $g \in G$ to the model rotates a value in channel c at position x to position $g \cdot x$ in channel $g \cdot c$, where $\cdot$ denotes the group rotation action. Based on the orbit structure, we can infer the reduced non-redundant space of weights. This does not restrict the expressivity of the model, and by optimizing each part of the model in its reduced non-redundant parameter space, we avoid the need to include rotated version of the data. The primary downside to this approach is greater implementation difficulty. There is an additional minor downside, which is the need to explicitly specify the orbit structure. In practice, for our experiments we always require all orbits to be isomorphic to the full invariance group G, which is guaranteed to impose no additional assumptions on the model but may skip some opportunities for removing redundant weights.

5.2 Experiments

We evaluated our approach on two datasets: a *Drosophila melanogaster* larval neuropil dataset collected using Focused Ion Beam Scanning Electron Microscopy (FIB-SEM) at a resolution of $10 \times 10 \times 10\,\mathrm{nm}$ voxels [68], and a rabbit retina dataset (referred to as E1088) collected using Serial Block Face Scanning Electron Microscopy (SBEM) at $22 \times 22 \times 25\,\mathrm{nm}$ voxel resolution [43].

Drosophila larva FIB-SEM dataset

On this dataset, we compared our approach against what was, at the time of our experiments, the state-of-the-art approach for segmentation of FIB-SEM datasets: [4]

1. A random forest classifier (RF1) is trained to detect boundaries using a small set of rotationally-invariant features computed using standard image processing operations,

such as gradient magnitude, Hessian eigenvalues, bilateral filtering, and structure tensor.

2. A seeded-watershed algorithm is used to obtain a supervoxel segmentation using the boundary predictions. A second random forest classifier (RF2) is trained to predict which adjacent supervoxels should be merged, using several size features as well as statistics collected along the face of some of the original features for RF1 as well as the output of RF1.

3. Finally, the algorithm computes the globally optimal segmentation that maximizes the product of face probabilities (based on RF2) subject to the multicut constraint (MC), which ensures consistency between merged faces.

Our comparison is based on source code posted by the author for the multicut-constrained optimization; the first two steps were reimplemented exactly according to the description in the paper.

This dataset consists of a single 500^3 voxel volume. For both approaches, we fixed one quarter of the volume as training data, one quarter of the volume as validation data, and the remaining half as test data.

The volume was manually annotated with a dense segmentation by human tracers, specified by on-voxel boundary labels with a uniform single-voxel thickness. [68] Because the precise voxel-level placement of boundaries is insignificant to intended uses of automatic segmentation, we dilated the ground-truth boundary up to 1 additional voxel in all directions (according to 26 connectivity) subject to topology preservation and only computed the error metrics outside of the dilated boundary.

For our sparse/wide approach, we used the complete symmetry group of all 48 axis-aligned rotations and reflections. We used $416 \cdot 48 = 19968$ $8 \times 8 \times 8$ clusters for the image patch k-means, $208 \cdot 48 = 9984$ $8 \times 8 \times 8$ clusters for the joint image and label patch k-means (which serve as the classes for the pooling features), and used $9 \times 9 \times 9$ patches of the primary features as input to the pooling layer. The final classifier used $16 \times 16 \times 16$ patches of both the primary and pooling features. We also evaluated a final classifier that uses only $16 \times 16 \times 16$ patches of the primary features, and does not use the pooling features. We performed a grid search over regularization parameters on the validation set. The best performing model (with and without the pooling features) was then evaluated on the test set. The segmentation accuracy on the test set, measured using Variation of Information and Rand Index, are shown in table 5.1. We also evaluated the effect of the amount of training data and number of first-layer k-means clusters on segmentation accuracy; the results are shown in fig. 5.2.

Rabbit retina SBEM dataset

This dataset consists of a 214^3 labeled voxel training volume, a 96^3 labeled voxel validation volume, and a 100^3 labeled voxel test volume. On this dataset, we compared our sparse/wide

Table 5.1: Results on the Drosophila larva FIBSEM test set using the optimal threshold. Both the Variation of Information (VI) and Rand Index (RI) are shown.

Approach	**VI**	**Rand index**
RF1	1.74201	0.975867
RF1+RF2	0.57070	0.995318
RF1+RF2+MC	0.45770	0.995620
Ours (Primary)	0.33565	0.999042
Ours (Primary+Pooling)	**0.32485**	**0.999134**

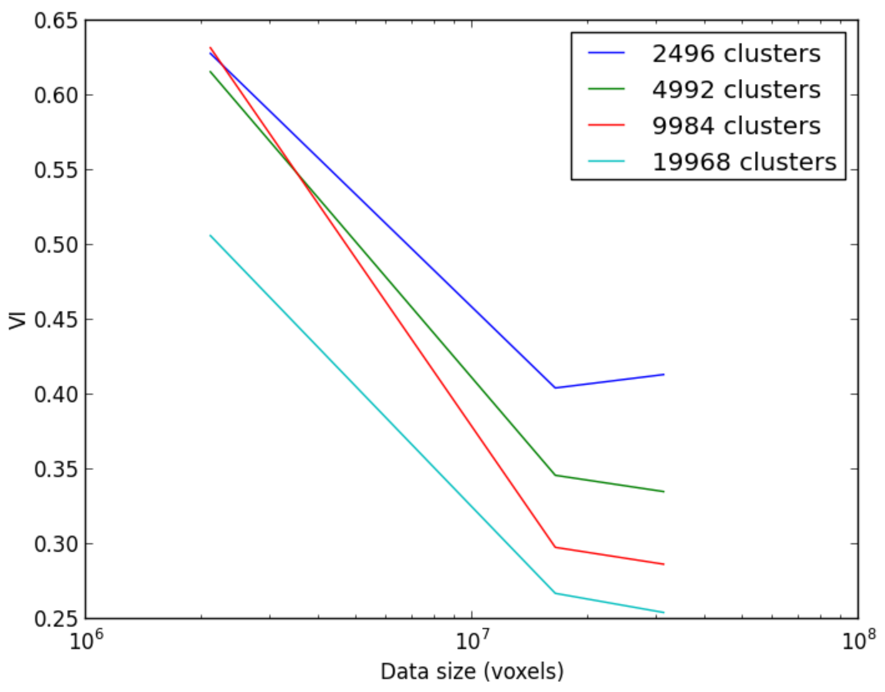


Figure 5.2: Effect of training data size and number of primary feature channels (k-means clusters) on validation performance, measured using Variation of Information (VI). Increasing the dimensionality of the primary feature representation significantly improve performance provided that there is sufficient training data, and even the largest feature dimensionality tested may not saturate performance. Increasing the amount of training data uniformly improves performance, and performance also does not appear to saturate even with the full amount of training data.

Table 5.2: Results on Rabbit retina SBEM test set. ($22 \times 22 \times 25\text{nm}^3$) $214 \times 214 \times 214$ voxel training set, $100 \times 100 \times 100$ voxel test set. The Rand Error, equal to 1 minus the Rand Index, is shown.

Method	Rand Error
Baseline	.0499
Boundary classification methods	
CN [50]	.0084
ilastik [78]	.0064
BLOTC CN [48]	.0056
MALIS CN [82]	.0055
Ours (Primary)	**.0036**
Agglomeration methods	
Single Linkage [49]	.0049
LASH [49]	**.0029**

approach to results for both pure boundary classification, as well as agglomeration applied on top of boundary classification, given in prior work for the same train/test split. [49]

For our sparse/wide approach, because of the anisotropic resolution of this dataset, instead of the full order 48 symmetry group used for the FIB-SEM dataset, we used the symmetry group of order 16 consisting of all 8 axis-aligned rotations and reflections in the x-y plane, combined with optional reflection in the z direction. We used $1248 \cdot 16 = 19968$ $8 \times 8 \times 8$ clusters for the image patch k-means, from which we computed the primary features. The final classifier used $16 \times 16 \times 16$ patches of the primary features. As for the FIB-SEM dataset, we performed a grid search over regularization parameters on the validation set.

The segmentation accuracy on the test set for each method is shown in table 5.2. In order to compare against the results given in prior work, we measured the accuracy of our method using Rand Error, equal to 1 minus the Rand index.

5.3 Discussion

On the Drosophila FIB-SEM dataset, our pure boundary classification approach, using just thresholding and connected components to compute the segmentation, significantly outperforms both the random forest-based boundary classification as well as the segmentation obtained by the state-of-the-art multicut-based agglomeration algorithm. Our pooling algorithm provides a small additional improvement. The very large number of parameters due to the very wide architecture of our model might seem to pose a risk of overfitting, but as shown in fig. 5.2, this does not seem to occur in practice; in fact, even with the smallest training subset, the largest model still performs best, and the improvement becomes larger as the amount of training data is increased.

On the SBEM dataset, our pure boundary classification approach outperforms all prior boundary classification approaches, including ones based on deep convolutional neural networks. It also exceeds the performance of the single linkage agglomeration, but does not quite match the performance of the LASH agglomeration method, which layers a sophisticated agglomeration approach on top of boundary classification. Although not tested in our experiments, methods like LASH could equally well be used on top of our boundary prediction.

A key advantage of our approach is that the training, based on clustering and batch optimization, is highly parallelizable, and can be completed in a matter of hours. In contrast, the deep neural network boundary classification methods to which we compared our method required weeks of training.

Because of the sparsity structure of our architecture, increasing the number of first-layer k-means clusters increases the computational cost of the feature computation, and increases the number of parameters, but the computational cost of the remainder of the approach does not increase. It is therefore quite feasible to increase the number of clusters even beyond what we used in our experiments.

Chapter 6

Correction of Inter-Section Discontinuities

6.1 Introduction

Recent technological developments in automated volume electron microscopy (EM) enable the acquisition of multi-terravoxel volumes at near isotropic resolution in the range of 3–30 nm [33, 40, 59, 74, 62]. These high-resolution image volumes are critical to fields such as connectomics, which aims to comprehensively map neuronal circuits by densely reconstructing neuron morphology and identifying synaptic connections between neurons [43, 41, 16].

All methods for volume electron microscopy, aside from tomography, which is only suitable for samples less than 1 micron in thickness, assemble the volume by spatially aligning a stack of two-dimensional images. Consequently, there can be substantial artifacts in the image volume, most notably serious discontinuities along the section axis. These are the result of variations in section thickness, sample deformations, and variations in imaging conditions. These artifacts are particularly a problem with ATUM-based SEM (Automated Tape Collecting Ultramicrotome-based Scanning Electron Microscopy) [74], a method that currently achieves the highest throughput and also has the advantage of preserving tissue sections, in contrast to the one-shot destructive imaging process of Serial Block Electron Microscopy (SBEM) [33] and Focused Ion Beam Scanning Electron Microscopy (FIBSEM) [59].

Figure 6.1 shows the artifacts typical in ATUM-based SEM volumes. In other imaging domains, improved imaging techniques and mathematical corrections have been devised for reducing artifacts in MRI and echo-planar images [38, 2, 1, 28], and in 2-D electron microscopy images [56], but there has been less focus on three-dimensional EM volumes. [58]

When present, these artifacts prevent automated and manual analysis of volumes except as a series of 2-D images along the original sectioning axis, preventing in particular the extraction of truly 3-D image features. Given that structures, such as neurites in cortex, may have arbitrary 3-D orientations relative to the original sectioning axis, this limitation is

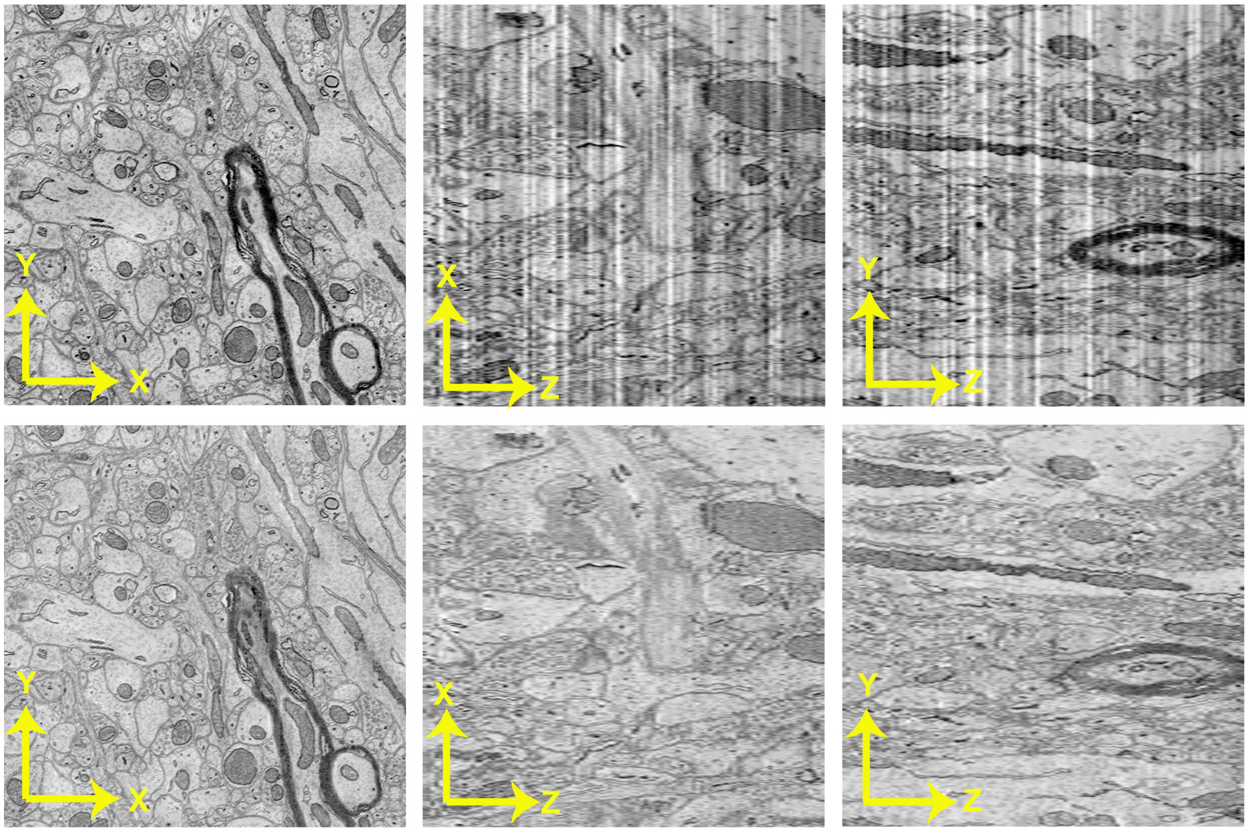


Figure 6.1: Representative cross-sectional views of a $6 \times 6 \times 30$ nm ATUM-based SEM [74] image volume of mouse cortex. *First row:* The original, registered dataset clearly showing discontinuities along the Z (section) axis. *Second row:* Corrected data using our EMISAC algorithm.

a serious impediment. On SBEM and FIBSEM volumes without these artifacts, the ability to view cross sections along arbitrary axes has aided humans tasked with manually tracing neurites and detecting synapses [43, 41], and automated algorithms for reconstructing neurite morphology (via segmentation) have depended on 3-D features. [50, 81, 48, 49, 5, 4]

To eliminate these artifacts and enable truly 3-D analysis of such image volumes, we propose a coarse-to-fine optimization-based procedure EMISAC (EM Image Stack Artifact Correction). We note that a single per-section brightness and contrast adjustment (i.e. linear transform of intensity values) is inadequate on typical datasets for correcting discontinuities except very locally. EMISAC optimizes the parameters of spatially varying linear transformations of the data in order to minimize the squared norm of the gradient along the section axis, subject to detail-preserving regularization, as described in section 6.2.

We applied EMISAC to a publicly available ATUM-based SEM volume of mouse cortex as well as to a serial section Transmission Electron Microscopy (ssTEM) volume of Drosophila larva ventral nerve cord. Figure 6.1 shows several cross-sectional views of the output of our algorithm. Qualitatively, EMISAC appears to completely eliminate all discontinuity artifacts while preserving all of the original detail.[1] (We did not attempt to address the problem of lower Z resolution than X-Y resolution.) Quantitatively, an evaluation based on the NIQE blind image quality metric [66] confirms the qualitative results. More importantly, we evaluated the effect of EMISAC as a preprocessing step on the accuracy of automated segmentation of neurites, a key challenge for this type of data; consistent with the NIQE scores, EMISAC dramatically improved segmentation accuracy.

After developing our approach independently, we became aware of a recent method [58] for addressing the same problem. Like our approach, this alternative method involves a quadratic optimization to minimize the squared norm of the gradient along the section axis, but uses a different parameterization and a different form of regularization and post-processing to preserve the intra-slice detail. We included this alternate method in our evaluations, and found EMISAC matches or outperforms it. In addition, we evaluated several other generic correction methods on the raw datasets including histogram equalization, contrast-limited adaptive histogram equalization [90], and local normalization, and found that EMISAC significantly outperforms all of them.

Furthermore, while designed primarily for electron microscopy image stacks, EMISAC is also applicable to lighting correction in time-lapse photography. Raw time-lapse image sequences typically have serious inter-frame lighting discontinuities. Although a few tools such as LRTimelapse[2] are designed to edit time-lapse sequences and correct these artifacts, our method does not require manual editing and can make local corrections to lighting as well. We evaluated EMISAC on several time-lapse videos exhibiting lighting problems; compared to the original videos, EMISAC produced a large qualitative improvement, and eliminated essentially all lighting discontinuities

This work was previously published. [6]

6.2 Artifact correction algorithm

In order to maximize the applicability of our approach, and avoid introducing a model bias[3] that could harm the accuracy of later stages of processing, we designed our approach to make as few and as simple assumptions as possible:

1. the true (undistorted) image volume is mostly continuous along the section axis;
2. the distortions in the volume can be expressed as *local* linear transformations of the intensity values, where the parameters of the linear transforms vary smoothly within each section (but are not smooth between sections).

[1] Our qualitative assessment was based on only a random subset of the cross-sections; we did not scrutinize every single cross-sectional view. Our quantitative assessment is more comprehensive.

[2] `http://lrtimelapse.com`

[3] A denoising method based on a learned sparse-coding dictionary, for instance, could potentially introduce patterns that were not present in the original data.

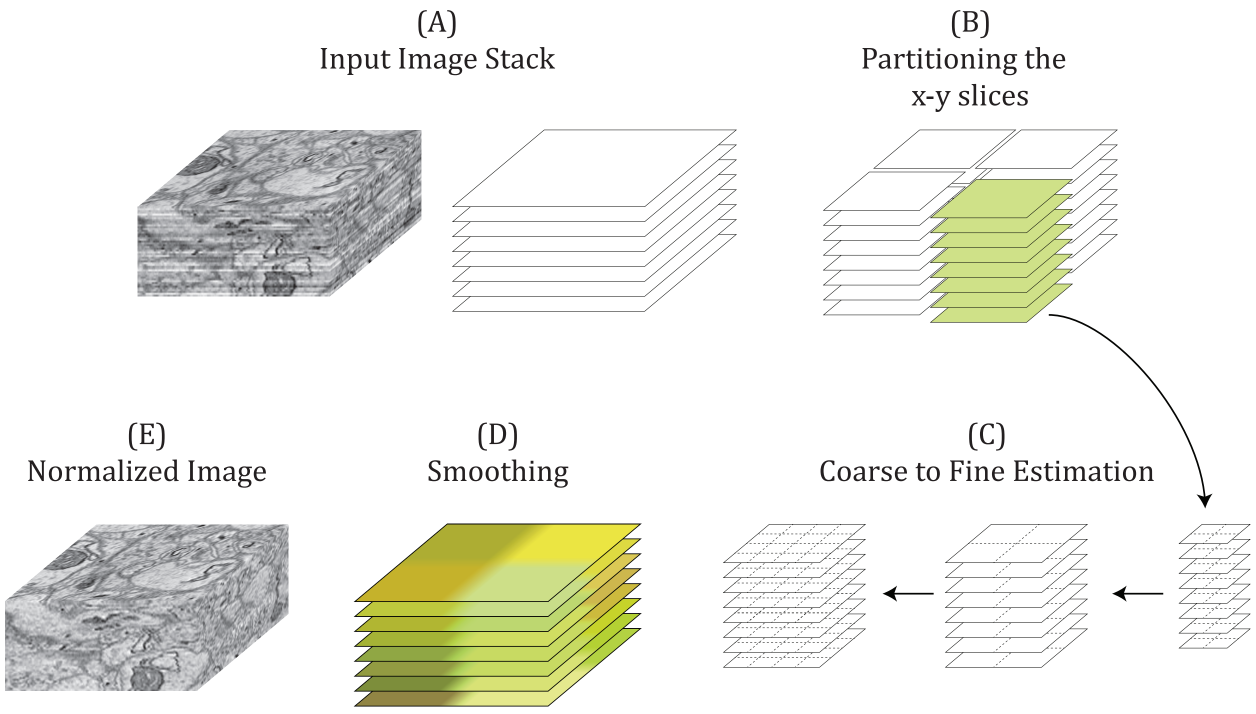


Figure 6.2: Schematic of our artifact correction approach. (A) An aligned EM image stack of 2-D slices. (B) The volume is optionally partitioned into a grid, which can be distributed across multiple processors; only limited communication is required between neighboring processors. (C) A coarse-to-fine procedure, in which both the data as well as the parameters are initially downsampled, speeds up the optimization of the objective in eq. (6.3). From right to left: downsampled x-y slices and downsampled parameters (larger block size); downsampled parameters only; full parameters (small block size). (D) The parameters are spatially smoothed within each x-y slice to remove the blocking effect. Four blocks are shown for illustration purposes, but in practice there are many more blocks. (E) Corrected output image.

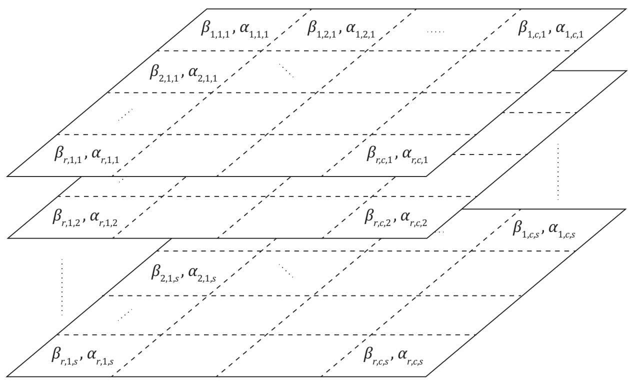
The detailed formulation of our method is explained in the following sections. A summary of our approach is illustrated in fig. 6.2.

Problem Formulation

As shown in fig. 6.3, we partition each slice of the 3-D EM volume into small fixed-size two-dimensional blocks. Voxel intensity values are corrected by a linear transformation given by:

$$I'_{x,y,z} = \beta_{\lfloor x/w_\mathrm{x} \rfloor, \lfloor y/w_\mathrm{y} \rfloor, z} \cdot I_{x,y,z} + \alpha_{\lfloor x/w_\mathrm{x} \rfloor, \lfloor y/w_\mathrm{y} \rfloor, z} \tag{6.1}$$

where $I(x, y, z)$ refers to the scalar intensity value at position (x, y, z) of the volume, and $(w_\mathrm{x}, w_\mathrm{y})$ is the block size. The smaller the block size, the larger the number of parameters.



$\beta_{1,1,1}, \alpha_{1,1,1}$
$\beta_{1,2,1}, \alpha_{1,2,1}$
$\beta_{1,c,1}, \alpha_{1,c,1}$
$\beta_{2,1,1}, \alpha_{2,1,1}$
$\beta_{r,1,1}, \alpha_{r,1,1}$
$\beta_{r,c,1}, \alpha_{r,c,1}$
$\beta_{r,1,2}, \alpha_{r,1,2}$
$\beta_{r,c,2}, \alpha_{r,c,2}$
$\beta_{1,c,s}, \alpha_{1,c,s}$
$\beta_{2,1,s}, \alpha_{2,1,s}$
$\beta_{r,1,s}, \alpha_{r,1,s}$
$\beta_{r,c,s}, \alpha_{r,c,s}$

Figure 6.3: EM image stack as a set of smaller blocks. The total number of blocks in the x-direction, y-direction, and z-direction are given by r, c, and s, respectively, resulting in a total of $r \cdot c \cdot s$ blocks. We have unique β and α parameters for each block.

Note that β and α correspond to correction of contrast and brightness, respectively. This block-based scheme effectively captures the local similarities within each slice and reduces the computational complexity.

We express our assumption of continuity in I' along z as a penalty on the squared norm of the gradient with respect to z. Likewise, the smoothness assumption on the the affine transforms is expressed as a penalty on the squared norms of the gradients of α and β with respect to the in-slice block positions. Thus, we formulate the optimization problem as follows:

$$\begin{aligned}\min_{\beta,\alpha} & \sum_x \sum_y \sum_z (I'_{x,y,z+1} - I'_{x,y,z})^2 \\ & + \gamma \sum_i \sum_j \sum_z \left[(\beta_{i+1,j,z} - \beta_{i,j,z})^2 + (\beta_{i,j+1,z} - \beta_{i,j,z})^2 \right. \\ & \qquad\qquad \left. + (\alpha_{i+1,j,z} - \alpha_{i,j,z})^2 + (\alpha_{i,j+1,z} - \alpha_{i,j,z})^2\right] \\ & \text{s.t.} \quad \beta_{i,j,z} \geq 1, \quad \forall i, j, z.\end{aligned} \tag{6.2}$$

The β parameters must be bounded to avoid the trivial null solution.

We reformulate the optimization problem given in eq. (6.2) as the convex quadratic problem

$$\min_X \|D_z X\|_2^2 + \gamma(\|D_x X\|_2^2 + \|D_y X\|_2^2) \qquad \text{s.t.} \quad \beta \geq 1, \tag{6.3}$$

where X=$[\beta, \alpha]$, $D_z = G_z A$, $D_x = G_x$, $D_y = G_y$, A is a matrix that maps the parameters X to a vector expressing I', and G_a is a matrix that maps a vectorized volume to a vector containing the finite-differences approximation of its gradient along axis a.

We use L-BFGS-B [87] to solve the optimization problem in eq. (6.3). The optimization is stopped when the fractional decrease in objective between consecutive iterations falls below

$\epsilon \cdot f$, where $\epsilon = 2^{-52}$ is the machine precision. To speed up the optimization process, we employ a coarse to fine estimation procedure detailed below.

Coarse-to-Fine Procedure

Rather than solving eq. (6.3) with the final desired block size directly, we solve a sequence of optimization problems of the form shown in eq. (6.3) with increasing image resolution in the x-y plane and/or decreasing block size. Each optimization after the first is initialized with the (appropriately upsampled) parameters that solved the previous optimization. In practice, we used (a single succession of) the following three steps:

1. The image resolution is reduced by a factor of 2 in x and y (using 2×2 averaging), as are the number of blocks.
2. The full image resolution is used, but the number of blocks remains reduced by a factor 2 in x and y.
3. The number of blocks is increased by a factor of 2 in x and y to its final size.

As the optimization is convex, the final solution obtained is unaffected by this procedure, but typically the running time is greatly reduced.

Removing the Blocking Effects

Ideally, the coarse-to-fine procedure is continued all the way down to a block size of $(1, 1)$. However, to reduce running time, it may be desirable to stop the procedure at a non-trivial block size. To avoid introducing artifacts from the blocking, after performing the final optimization, we upsample the α and β parameters to a block size of 1 using linear interpolation (rather than nearest neighbor interpolation).

Parallelization

For large image volumes, we can partition the volume into a grid along the x and y axes, which can be distributed across multiple processors or machines. For simplicity, the grid cell boundaries should be aligned to block boundaries. Only the parameter gradients for blocks on the border of each grid cell must be communicated, at each iteration of the optimization, to the processors responsible for neighboring grid cells, which is in general a very small amount of data relative to the size of the image volume. As a simplifying approximation, we could even ignore the regularization term between neighboring grid cells and thereby require no communication between machines. In practice this may not significantly affect the result provided that the grid cells are large enough. In our experiments, we observed no loss of accuracy (in NIQE score) from using no communication between blocks, except for the final upsampling step.

6.3 Evaluation on electron microscopy data

We tested EMISAC on a publicly available ATUM-based SEM volume of mouse cortex released by Kasthuri *et al.* [54] For a $1024 \times 1024 \times 100$ voxel portion of this dataset, a dense segmentation of neurites traced by a human expert is also publicly available as part of the SNEMI3D neurite segmentation challenge [9], which enabled us to evaluate the effect of EMISAC on segmentation accuracy. The dataset was acquired at a resolution of $3 \times 3 \times 30$ nm, but as the human-traced segmentation is provided only at the downsampled resolution of $6 \times 6 \times 30$ nm, we used the same downsampled resolution for all of our experiments. In addition, we tested EMISAC on another publicly available dataset, Cardona *et al.* 2010, collecting using a different imaging technique, ssTEM rather than ATUM-SEM, and of Drosophila larva ventral nerve cord rather than mouse cortex, with a resolution of $4 \times 4 \times 50$ nm [20, 21].

We compared EMISAC against the original aligned but uncorrected image volume. We are aware of only one other method designed to address this same problem (of which we only became aware after independently developing our own algorithm), which we refer to as Kazhdan2013 [58]. As the authors of that method made publicly available their output [57] on the same mouse cortex volume on which we tested EMISAC, we were able to include the Kazhdan2013 algorithm in our evaluations without having to reimplement it. We also compared EMISAC against histogram equalization, contrast-limited adaptive histogram equalization (CLAHE) [90], and local normalization [73].

For EMISAC, we set the affine transform regularization parameter $\gamma = 0.4\frac{w_\mathrm{x} w_\mathrm{y}}{d_\mathrm{x} d_\mathrm{y}}$ for all electron microscopy datasets, where d_x and d_y are the image downsampling factors in x and y respectively (relative to the $6 \times 6 \times 30$ and $4 \times 4 \times 30$ nm resolution data). The coefficient was selected to minimize NIQE score, which does not depend on any labeled data. Furthermore, results were fairly insensitive to several orders of magnitude change in γ.

Image quality evaluation

Although any corrected version of the data is only useful in so far as it aids an image analysis task of interest, such as neurite segmentation, it is convenient to be able to directly quantify the image quality independent of any particular later analysis step. A visual assessment by humans would be inherently subjective (and also inconvenient), and it is impossible to obtain a "ground truth" version of the data without any imaging artifacts, against which the corrected version might be compared. Furthermore, we have no way of knowing the true distortion model. We therefore rely on the "completely blind" Natural Image Quality Evaluator (NIQE) [66], which requires neither a model of expected distortions nor human assessments of distorted images as training data, but merely a set of high quality images from which to estimate a model of natural image statistics. On natural image benchmarks, this method is comparable to the best methods that *do* rely on human assessments as training data.

We trained a NIQE model on a random set of x-y sections from the dataset that were not part of any of the volumes on which we tested our approach. Thus, the NIQE score of a y-z or x-z cross section represents the statistical similarity of the orthogonal cross sections to the original image sections, a very reasonable metric given that the ultimate goal is to be able to analyze the 3-D volume without regard to a preferred orientation.

Segmentation accuracy evaluation

As one of the primary goals of our work on the artifact correction is the improvement of automated segmentation results for neural circuit reconstruction, we directly evaluated the impact of EMISAC on 3-D segmentation accuracy. The training of our machine-learning-based segmentation algorithm, as well as the evaluation of segmentation accuracy, were based on the SNEMI3D human expert-traced segmentation, which we treated as "ground truth."

We used a three-step segmentation procedure, based on the sparse wide boundary classification approach described in chapter 5:

1. We use an unsupervised procedure to transform each position in the image volume into a 1-of-k binary feature vector, with $k = 624 \cdot 16$. We cluster $16 \times 16 \times 4$ patches of the image volume using k-means based on L_1 distance; the binary feature vector for each position is obtained by vector quantizing the image patch centered at that location. We take advantage of the assumed rotational covariance of the data (namely transposition and reflection in the x-y plane, and reflection along the z axis), which reduces the effective number of parameters by a factor of 16.

2. To predict the presence of a cell boundary between two adjacent voxels along the x, y, or z axes, we train a logistic regression classifier for each of the 3 axes. The feature vector for classification is obtained by concatenating all of the 1-of-k binary feature vectors within a $16 \times 16 \times 16$ window around the boundary (producing a very high-dimensional feature vector). We extracted boundary information for training examples directly from the human-provided ground-truth segmentation, and ensured that equal total weight was given to positive and negative training examples. We optimized the classification model using L-BFGS, using a quadratic approximation to dropout [84] (with $p = 0.5$) for regularization. As for the unsupervised feature learning, we take advantage of the assumed rotational covariance to reduce the number of parameters to be learned by a factor of 16.

3. To produce a segmentation, we employ Gala [68], a state-of-the-art electron microscopy image segmentation algorithm, based on agglomeration of supervoxels, for which we use the cell boundary predictions as input.

We use half of the ground truth segmentation to train the boundary classifier, half of the remaining portion to train Gala, and the remainder for evaluation. The same training/testing

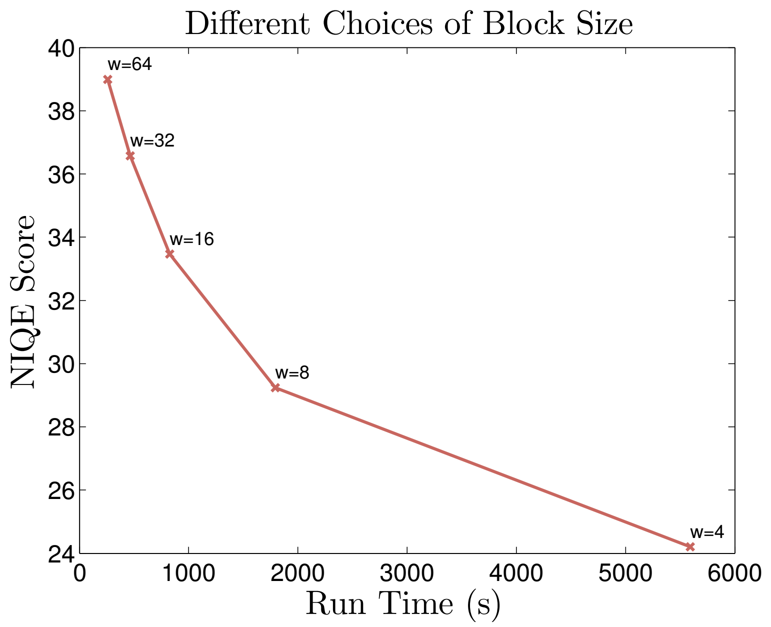


Figure 6.4: Plot of average NIQE score versus EMISAC running time on the $1024 \times 1024 \times 100$ voxel SNEMI3D portion of the mouse cortex volume. The NIQE scores for all x-y, x-z, and y-z cross sections within the volume are averaged. w stands for the block size of (w, w). A lower NIQE score corresponds to higher image quality. The optimization was run in all cases with a stopping threshold of $f = 10^{10}$. These NIQE scores are consistent with the higher rate of visually apparent artifacts present when larger block sizes are used, which provides some confirmation of the validity of the NIQE score.

procedures were used for the original data, the EMISAC output, and the Kazhdan2013 output. The results are averaged over 4 splits.

6.4 Electron microscopy results

To guide later experiments, we initially evaluated the effect of varying the block size on running time and image quality (measured by NIQE score); the results are shown in fig. 6.4. The running times reported for this and later experiments are based on our Python implementation running on an 8-core Intel Xeon X5570 2.93 GHz system, which consumed about 15 GB memory for each $1024 \times 1024 \times 100$ volume. Based on the observed trade-off between image quality and running time, we used a final block size of $(16, 16)$ for later experiments. We found that the NIQE score typically converged before reaching the threshold of $f = 10^{10}$.

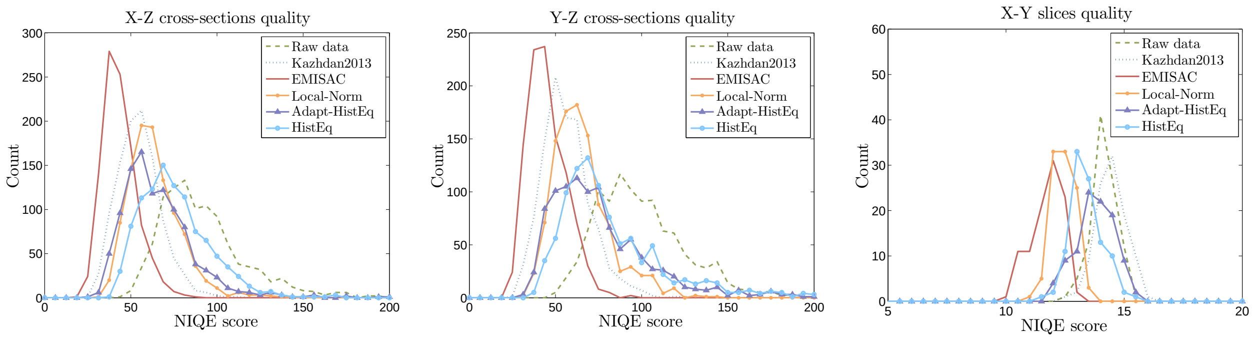


Figure 6.5: Histogram of NIQE scores for the SNEMI3D volume. EMISAC uses a block size of $(16, 16)$ and stopping criteria of $f = 10^{10}$. Lower scores are better.

Comparison of NIQE scores

Using these parameters, we evaluated the improvement in NIQE score relative to the original data of EMISAC, Kazhdan2013, histogram equalization, contrast-limited adaptive histogram equalization [90], and local normalization. Figure 6.5 shows the distribution of NIQE scores for the SNEMI3D volume. Table 6.1 shows the improvement in NIQE score on both datasets. Under Welch's t-test, EMISAC attains a large and highly statistically significant improvement in both x-z ($p < 0.0001$) and y-z ($p < 0.0001$) NIQE score relative to Kazhdan2013 on the ATUM-SEM volume, without any considerable loss in x-y NIQE score. The preservation of detail is confirmed by the very high structural similarity (SS) [85] between the original and corrected x-y cross-sections. See fig. 6.6 for a visual comparison.

Comparison of Segmentation Accuracy

For the original data and each correction method, we evaluate the segmentation accuracy on each of the 4 boundary training/agglomeration training/test splits of the SNEMI3D volume. The Gala segmentation algorithm has a threshold parameter that trades off between false merges and false splits, as shown in fig. 6.7; the optimal trade-off depends on the particular application, but in order to summarize results, we simply compute the minimum VI score (which gives equal weight to false splits and merges) over all thresholds.[4] For each correction method on each split, we compute the percent decrease in *minimum* VI score relative to the original data. To compare methods, we compute the mean and standard deviations of these decrease percentages. The results are shown in table 6.2.

The nearly exact match in x-y NIQE scores between the original data and Kazhdan2013, as shown in fig. 6.5 and table 6.1, can be explained by the fact that Kazhdan2013 essentially

[4]In actual use, we would have to pick the threshold based on cross-validation, as there would be no way to determine the true VI score for each threshold. However, this added complexity is irrelevant to our evaluation of artifact correction methods.

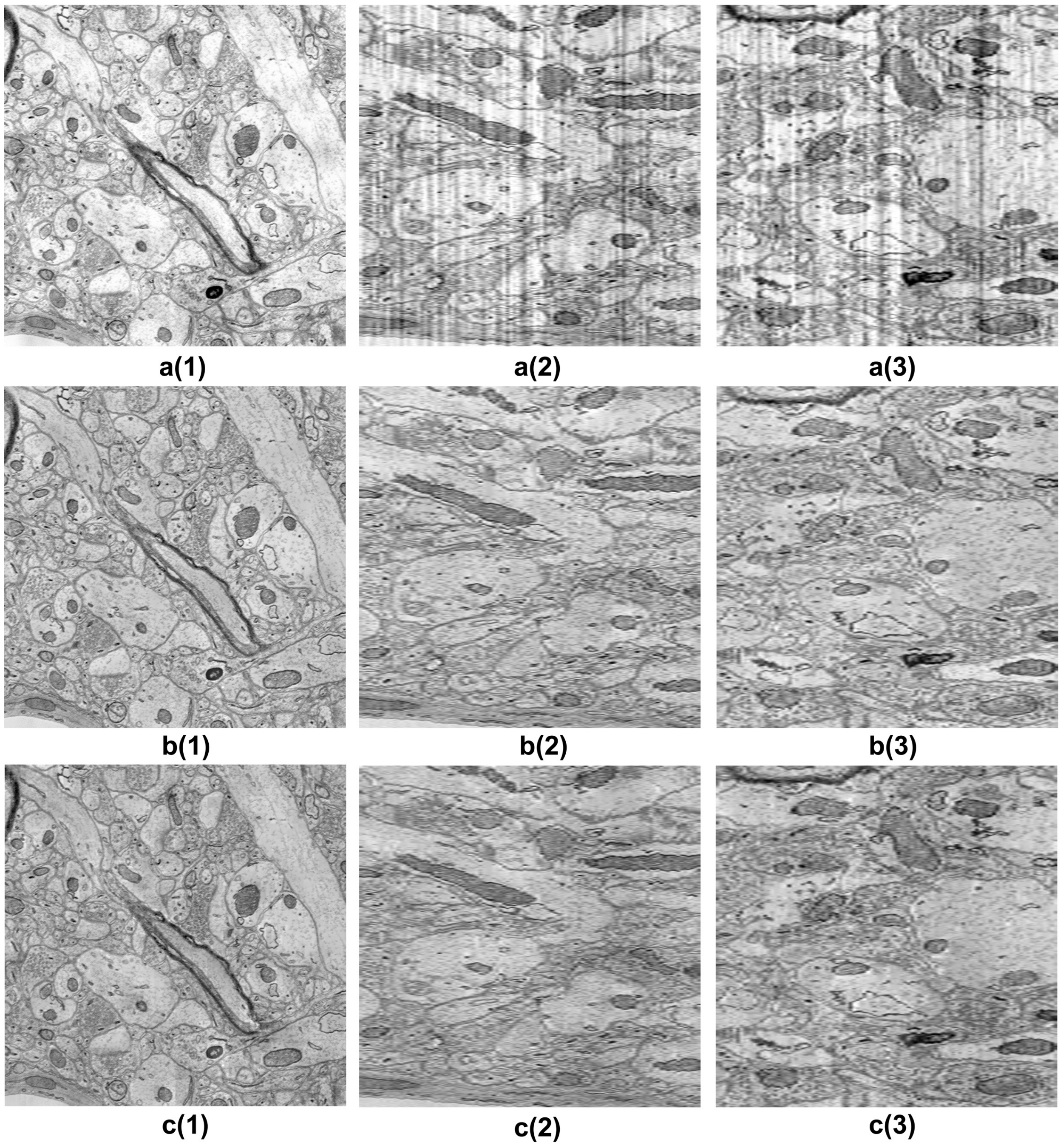


Figure 6.6:]
Visual comparison of (a) the original data, and the corrected versions using (b) Kazhdan2013 and (c) our method EMISAC. From left to right we have the (1) x-y cross section, (2) x-z cross section, and (3) y-z cross-section. The two correction algorithms produce visually very similar results.

Table 6.1: NIQE score reduction (improvement) percentage, averaged over all x-y, x-z, and y-z cross-sections in the two EM datasets. The average structural similarity index (SS) [85] (as a percentage) between the original and corrected x-y cross-sections is also shown. Higher percentages are better. *First column:* Six 1024 × 1024 × 100 voxel volumes of the mouse cortex ATUM-SEM volume [54] (five randomly sampled volumes plus the SNEMI3D volume). *Second column:* 512 × 512 × 30 *Drosophila* larva ventral nerve cord ssTEM volume [21].

	Mouse cortex volume				Drosophila larva ventral nerve cord volume			
	x-y	x-z	y-z	SS	x-y	x-z	y-z	SS
EMISAC	6.59 ±6.86	**41.21** ±19.25	**39.56** ±19.82	96.4 ±2.20	-1.15 ±2.83	**11.04** ±25.05	**11.82** ±23.27	97.4 ±1.06
Kazhdan2013	-1.85 ±0.82	35.24 ±20.28	35.06 ±20.02	98.3 ±1.69	-	-	-	-
Histogram Equalization	-4.53 ±6.02	-35.80 ±86.31	-42.31 ±95.04	69.5 ±6.47	-21.16 ±17.79	-68.54 ±88.76	-67.92 ±84.56	81.0 ±3.64
CLAHE	-7.59 ±6.19	-39.77 ±95.09	-40.46 ±96.56	70.3 ±3.73	-14.02 ±9.43	-127 ±150	-133 ±154	80.1 ±3.76
Local Normalization	2.51 ±4.91	13.04 ±27.14	13.79 ±27.20	93.3 ±3.52	3.05 ±1.93	-6.38 ±26.72	-8.51 ±28.85	97.4 ±1.58

copies the high-frequency content of the original x-y slices in its final step, and NIQE scores depend only on local (high-frequency) information.

While the NIQE scores were relatively insensitive to the stopping threshold f, we observed that the segmentation accuracy was highly sensitive, and therefore computed results for $f \in \{10^{10}, 10^9, 10^7, 10^6\}$, corresponding to increasing segmentation accuracy. Both Kazhdan2013 and EMISAC (for $f \leq 10^9$) achieve a similarly large improvement in accuracy over the original data. The difference between the two methods is not statistically significant ($p = 0.87$).

Figure 6.4 suggests that better results may be possible by using a block size smaller than $w = (16, 16)$, which was chosen for convenience in running experiments given the speed of our implementation.

We report running times of our Python implementation for comparison purposes, but by no means expect them to be comparable to those of a highly-optimized CPU or GPU implementation. Furthermore, L-BFGS-B is by no means the most effective algorithm for optimizing eq. (6.3). The focus of our work was in evaluating correction models, rather than implementation speed.

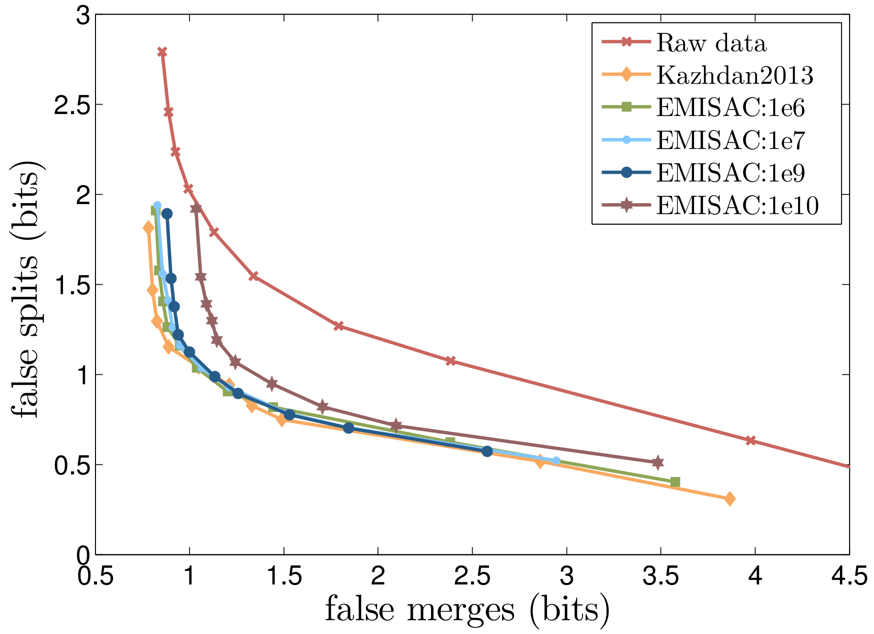


Figure 6.7: Plot of the $H(S|U)$ (false split) vs. $H(U|S)$ (false merge) trade-off for segmentations based on the original image volume, Kazhdan2013, and EMISAC with several values of the stopping criteria f. Lower scores are better. Results are shown just for a single split, but results on other splits are similar. Note that the variation of information is simply $H(S|U) + H(U|S)$.

Table 6.2: Effect of artifact correction on segmentation accuracy ($n = 4$)

	Kazhdan2013	EMISAC			
		1e10	1e9	1e7	1e6
VI improvement(%)	29.19	19.83	26.43	27.23	27.97
	±13.12	±11.24	±10.03	±2.79	±1.88
Run Time (s)	-	828	1847	3963	4874

Convolutional neural networks have shown good performance for neurite boundary detection [82, 48, 23], and may well perform better than the boundary classification method we used for our segmentation evaluation. Our choice was motivated by the fact that the state-of-the-art 3-D convolutional neural network approach for this problem is currently far from a settled matter, and we believe our method to be similar in performance; furthermore, it would have been highly impractical to spend the several weeks to months[5] of GPU time required to the train the network for each variant and data split that we tested.

[5]State-of-the-art 2-D networks often require several weeks of GPU training [23, 60]; a comparable 3-D network can be expected to take at least as long, and possibly several times longer due to the larger number

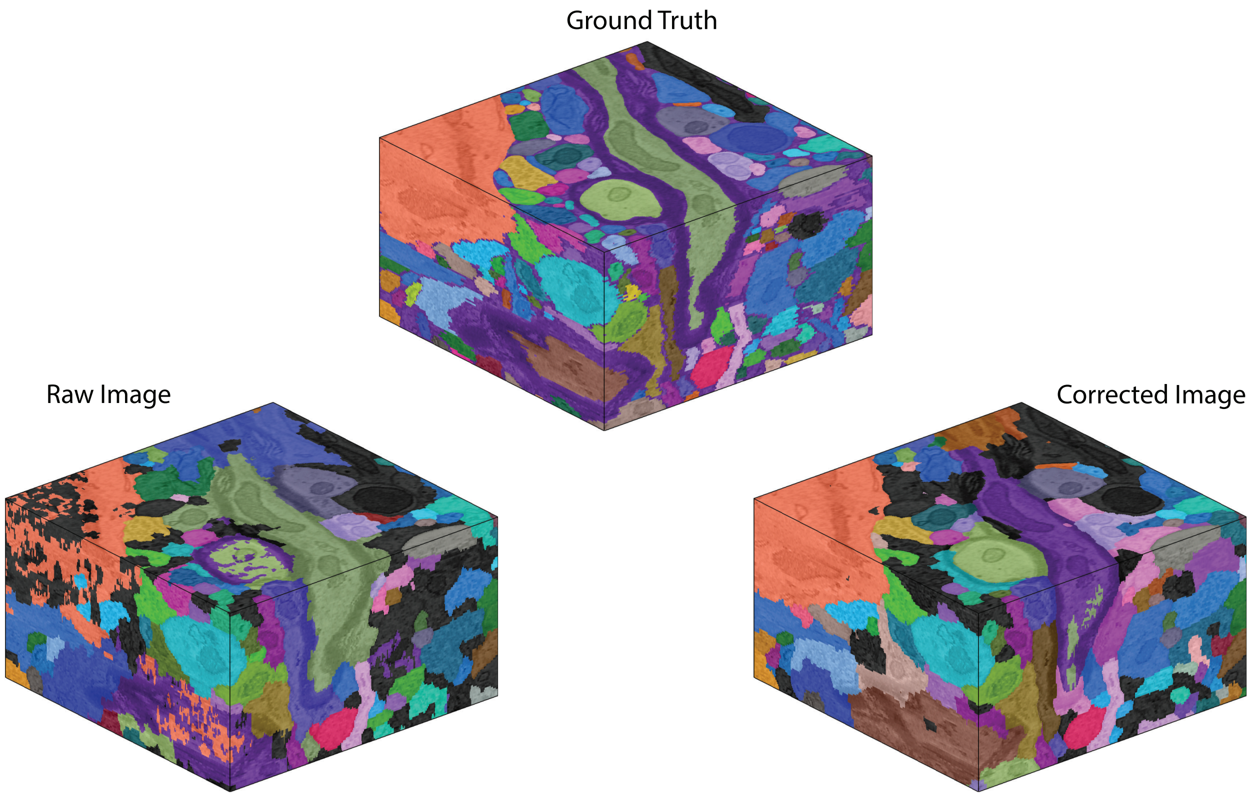


Figure 6.8: Qualitative improvement in segmentation accuracy from EMISAC.

6.5 Lighting correction of time-lapse photography

Raw time-lapse photography sequences typically exhibit substantial flickering due to variations in lighting and exposure between frames [22]. Due to scene geometry, these variations are often local, such that a global brightness and contrast adjustment per frame is insufficient. We can directly apply EMISAC to the problem of correcting such lighting issues by treating time as the z-axis (taking the place of the section axis for the EM data); a separate set of α and β parameters are used for the red, green, and blue channels.

Quantitative evaluation of these time-lapse sequences cannot be done in the same way as for electron microscopy stacks, since the data distribution is obviously not invariant to transpositions between time and the x or y axis, as would be implied by comparing x-z and y-z cross-sections to the original x-y frames. In fact, we are unaware of any established method for quantitatively measuring lighting discontinuity in time-lapse sequences; while EMISAC's own objective function does measure this in some sense, it cannot reasonably be used for comparison to other methods nor can it be aggregated across datasets. Therefore, we are limited to qualitative assessment.

of parameters.

For all time-lapse sequences, we used a final block size of $(w_x = 4, w_y = 4)$ and manually set γ in the range of $[\frac{39\times10^2 w_\mathrm{x} w_\mathrm{y}}{d_\mathrm{x} d_\mathrm{y}}, \frac{39\times10^4 w_\mathrm{x} w_\mathrm{y}}{d_\mathrm{x} d_\mathrm{y}}]$; γ trades off preservation of detail and temporal smoothness, and is fundamentally a matter of user preference.

For evaluation, we used 8 publicly available time-lapse sequences that exhibited lighting discontinuities between frames. For several of these sequences, a demonstration result obtained by manual editing using the commercial LRTimeLapse software was also available. The corresponding video files can be found at `http://rll.berkeley.edu/2014_ECCV_EMISAC`. Qualitatively, EMISAC essentially eliminates all flickering without reducing the apparent quality of individual frames. It appears to give a very similar quality result, in terms of correcting lighting discontinuities, to that obtained by manual editing with the specialized LRTimeLapse software.

6.6 Discussion

Imaging artifacts, most notably discontinuities along the section (z) axis, have so far limited the use of image volumes acquired by ATUM-based SEM, one of the most promising high-throughput volume electron microscopy techniques, to essentially 2.5-D analysis [83, 23]. Our limited assumptions about the data and distortion process lead naturally to a simple but highly effective optimization-based procedure: our method EMISAC appears to eliminate all visible discontinuities, without any loss of intra-section detail. On the key task of neurite segmentation, EMISAC substantially improves accuracy relative to the original data by about 28%, matching the improvement achieved by the recent independently-developed alternative method Kazhdan2013 [58]. In terms of NIQE score [66], our method significantly outperforms Kazhdan2013. Furthermore, the significant qualitative improvement in the video results demonstrates the applicability of EMISAC to time-lapse photography.

One explanation for the superior NIQE scores attained by EMISAC compared to Kazhdan2013 may be that EMISAC supports both local brightness as well as local contrast correction. Kazhdan2013 solves two sequential optimization problems, both of which apply an additive correction term to the original data, and penalize deviations in intra-slice gradients of the correction term. This implicitly allows smooth changes in brightness, as the gradient of the correction term is only affected by the gradient of the brightness factor. Even constant changes in contrast, however, are penalized heavily, as they have a *multiplicative* effect on the gradient of the correction term. While the lower NIQE scores of EMISAC relative to Kazhdan2013 did not correspond to better segmentation accuracy in our experiments, the segmentation performance may have been limited by the quality of the feature representation, and a future segmentation algorithm may indeed show improvement.

Chapter 7

Combinatorial Energy Learning

As discussed in chapter 3, algorithmic approaches to image segmentation are often formulated as a variation of the following pipeline: a boundary detection step that establishes local hypotheses of object boundaries, a region formation step that integrates boundary evidence into local regions (i.e. superpixels or supervoxels), and a region agglomeration step that merges adjacent regions based on image and object features. [5, 49, 83, 4] Although extensive integration of machine learning into such pipelines has begun to yield promising segmentation results [13, 37, 68], we argue that such pipelines, as previously formulated, fundamentally neglect two potentially important aspects of achieving accurate segmentation: (i) the combinatorial nature of reasoning about dense image segmentation structure,[1] and (ii) the fundamental importance of shape as a criterion for segmentation quality.

Contributions: We propose a method that attempts to overcome these deficiencies. In particular, we propose an energy-based model that scores segmentation quality using a deep neural network that flexibly integrates shape and image information: Combinatorial Energy Learning for Image Segmentation (CELIS). In pursuit of such a model this paper makes several specific contributions:

- a novel connectivity region data structure for efficiently computing the energy of configurations of 3-D objects.
- a binary shape descriptor for efficient representation of 3-D shape configurations.
- a neural network architecture that splices the intermediate unit output from a trained convolutional network as input to a deep fully-connected neural network architecture that scores a segmentation and 3-D image.
- a training procedure that uses pairwise object relations within a segmentation to learn the energy-based model.

[1]While prior work [5] has recognized the importance of combinatorial reasoning, the method proposed only addressed it to a limited extent.

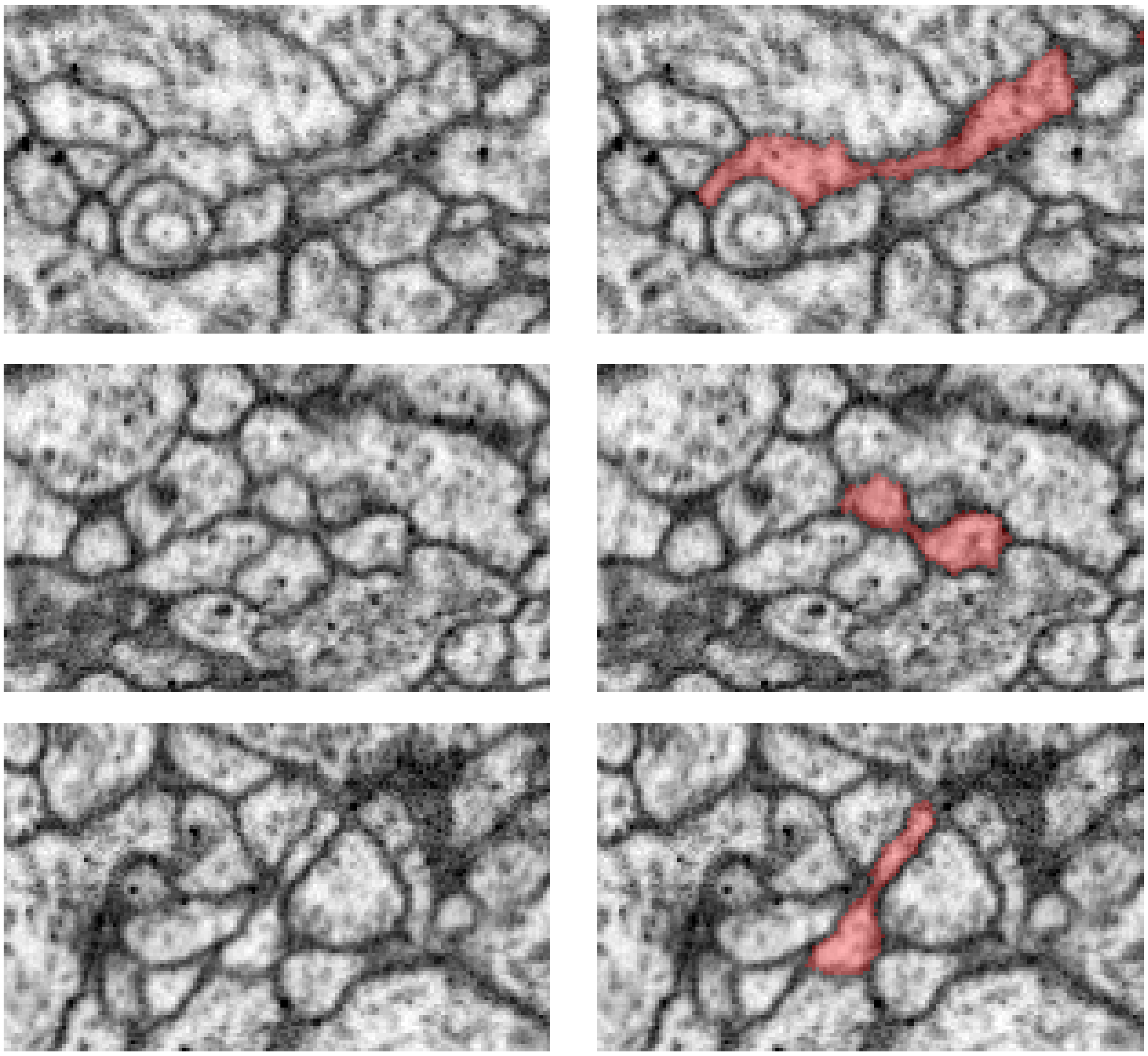


Figure 7.1: Examples of cases where local boundary classification alone leads to false splits of neurites. A cross-section of the raw data is shown on the left; the correct segmentation (determined by careful human annotators) of the central neurite is overlayed on the right. Neuronal processes often narrow to nearly the limit of the image resolution, and when this is coupled with a loss of contrast, it appears to be impossible to determine the correct segmentation from local boundary information alone. These examples are from a Drosophila larval neuropil dataset [68] imaged using Focused Ion Beam Scanning Electron Microscopy (FIBSEM) [68].

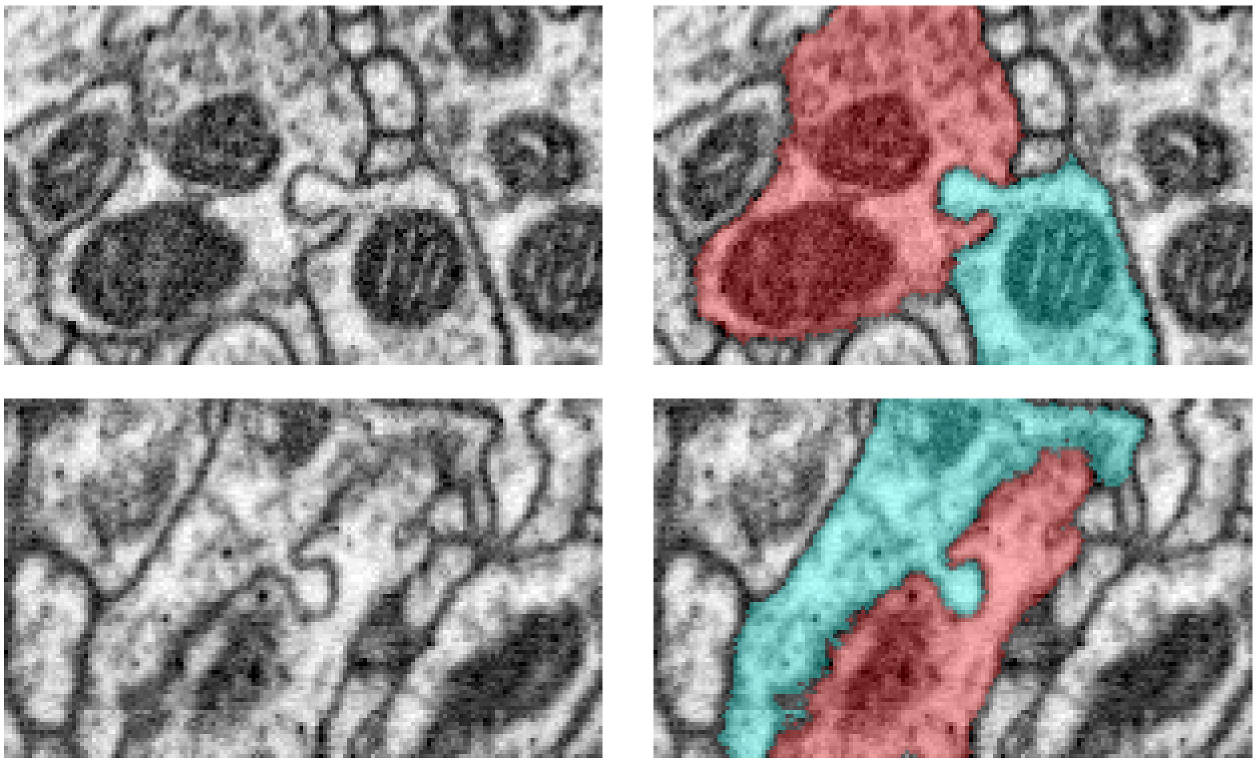


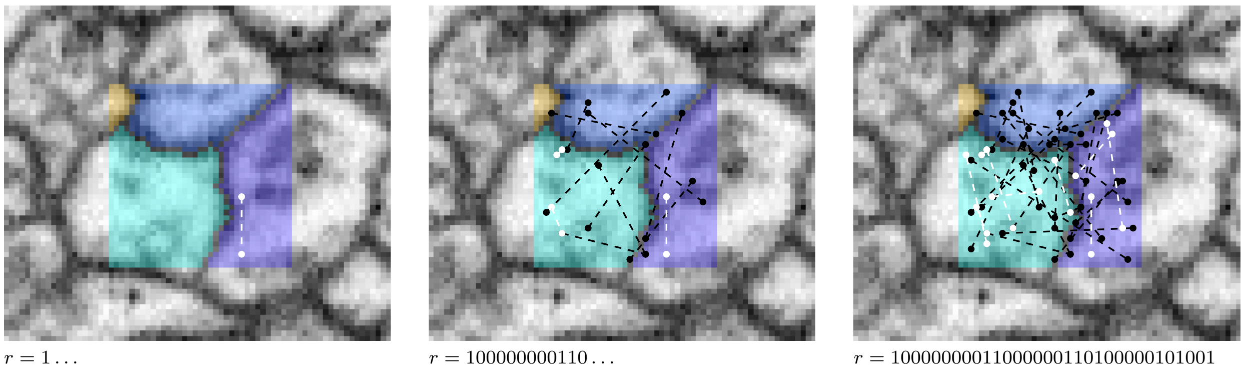
Figure 7.2: Examples of cases where independent neurite shape modeling breaks down. At these synapse sites, the pre-synaptic and post-synaptic neurons each have characteristic shapes that are highly unlikely to occur independently but are jointly very likely. Due to the close contact between the two neurons, local boundary classification at these sites often results in false mergers, making correct shape modeling particularly critical. A cross-section of the raw data is shown on the left; the correct segmentation (determined by careful human annotators) is overlayed on the right. These examples are from a Drosophila larval neuropil dataset [68] imaged using Focused Ion Beam Scanning Electron Microscopy (FIBSEM) [68].

On two challenging datasets of 3-D connectomics data, we show the proposed approach can be learned from data, and can be used to improve topological reconstruction quality by scoring a sequence of manipulations applied to a segmentation.

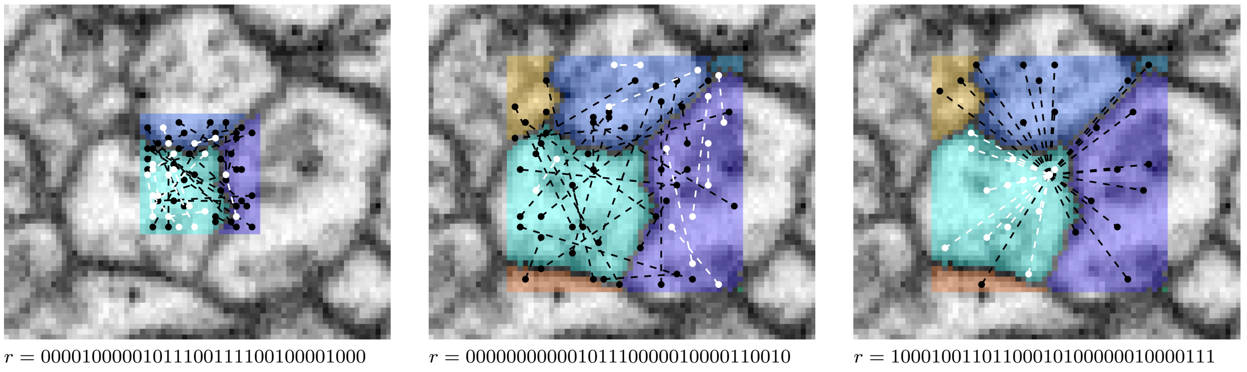
7.1 Representing 3-D shape configurations with local binary descriptors

We propose a *binary shape descriptor* based on subsampled pairwise connectivity information.

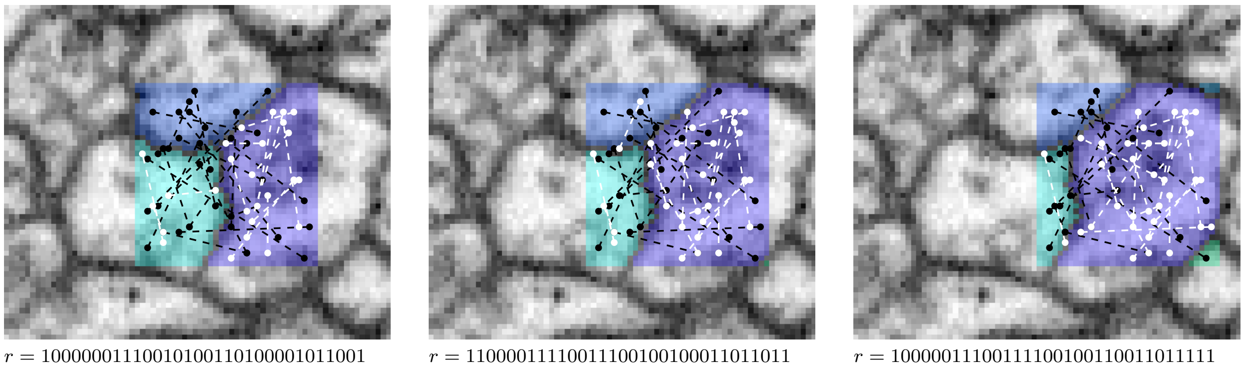
Definition 7.1. A k-bit *binary shape descriptor specification* s is a set of k distinct pairs



(a) Sequence showing computation of a shape descriptor.


$r = 000010000001011100111100100001000$
$r = 000000000001011100000010000110010$
$r = 100010011011000101000000010000111$

(b) Shape descriptors are computed at multiple scales. Pairwise descriptors (shown left and center) consider arbitrary pairwise connectivity, while center-based shape descriptors (shown right) restrict one position of each pair to be the center point.


$r = 100000011100101001101000001011001$
$r = 110000111100111001001000110110 11$
$r = 100000111001111001001100110111111$

(c) Shape descriptors are computed densely at every position within the volume.

Figure 7.3: Illustration of shape descriptors. The connected components of the bounding box U for which the descriptor is computed are shown in distinct colors. The pairwise connectivity relationships that define the descriptor are indicated by dashed lines; connected pairs are shown in white, while disconnected pairs are shown in black. Connectivity is determined based on the connected components of the underlying segmentation, not the geometry of the line itself. While this illustration is 2-D, shape descriptors are in general computed in 3-D.

of position offsets $\{\{a_1, b_1\}, \ldots, \{a_k, b_k\}\}$ relative to the center of a bounding box of some fixed-size B_s (specified as a 3-D vector of integers).

For simplicity, we will require that all components of B_s are odd, to allow shape descriptors to be exactly centered on single voxels.

Definition 7.2. Given a binary shape descriptor specification $s = \{\{a_1, b_1\}, \ldots, \{a_k, b_k\}\}$, the corresponding k-bit *binary shape descriptor* r of a segmentation U (of size B_s) is defined by

$$r^i(S) := \begin{cases} 1 & \text{if } a_i \text{ is connected to } b_i \text{ in } U; \\ 0 & \text{otherwise.} \end{cases} \qquad \text{for } i \in [1, k].$$

As shown in fig. 7.3a, each bit of the descriptor specifies whether a particular pair of positions are part of the same segment. By using a suitable data structure that maintains connected component information, each bit can be computed in constant time. Note that the ordering of the bits is arbitrary, and conceptually we can consider each bit to be indexed by the pair $\{a, b\}$ of position offsets to which it corresponds.

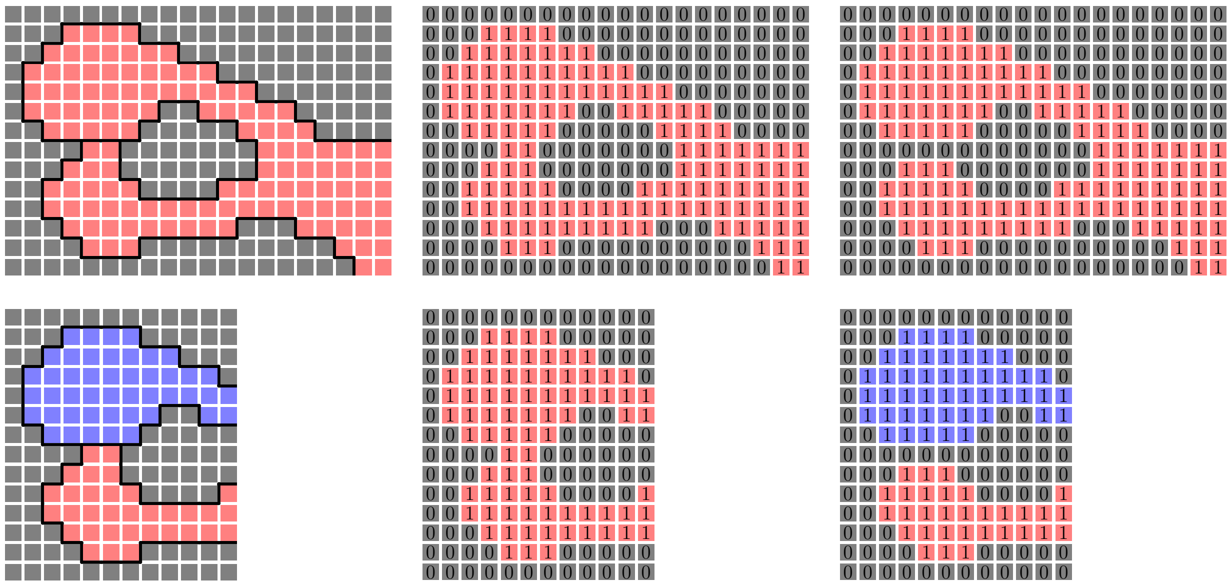
In the limit case, if we use the set of all $k = \binom{n}{2}$ pairs of positions within an n-voxel bounding box, no information is lost and the Hamming distance between two descriptors is precisely equal to the Rand index. [70] In general we can sample a subset of only k pairs out of the $\binom{n}{2}$ possible; if we sample uniformly at random, we retain the property that the *expected* Hamming distance between two descriptors is equal to the Rand index. The representation is therefore highly sensitive to topological changes that affect connectivity, but is fairly insensitive to small perturbations that do not change the topology, which is exactly what is desired for a robust representation.

We found that picking $k = 512$ bits provides a reasonable trade-off between fidelity and representation size. While the pairs may be randomly sampled initially, to obtain consistent results when training models based on these descriptors we must use the same fixed list of positions for defining the descriptor at both training and test time.

Note that this descriptor serves in general as a type of sketch of a full segmentation of a given bounding box. By restricting one of the two positions of each pair to be the center position of the bounding box, we instead obtain a sketch of just the single segment containing the center position. We refer to the descriptor in this case as *center-based*, and to the general case as *pairwise*, as shown in fig. 7.3b. We will use these shape descriptors to represent only *local sub-regions* of a segmentation. To represent shape information throughout a large volume, we compute shape descriptors densely at all positions in a sliding window fashion, as shown in fig. 7.3c.

7.2 Connectivity regions

As defined, a single shape descriptor represents the segmentation within its fixed-size bounding box; by shifting the position of the bounding box we can obtain descriptors corresponding



(a) Graph representation (b) Component representation (c) Component representation

Figure 7.4: Advantage of voxel graph representation. The top row shows a representation of a segmentation as either a voxel graph or a component labeling. The bottom row shows the effect of restricting the segmentation to a sub-region. Each square corresponds to a voxel. In the graph representation, a white line between two voxels indicates an edge, while a black line indicates the lack of an edge. In the component representation, each voxel is labeled by a component identifier (0 or 1). The different colors (red, blue, and grey) correspond to different connected components. The graph representation, shown on the left, correctly disconnects the two parts when restricted to the sub-region. The component labeling representation, shown in the middle, is unable to represent the presence of a boundary between the two parts, and therefore incorrectly results in a single connected component even when restricted to the sub-region. It is possible to emulate a voxel graph using a component representation by indicating boundaries with a 1-voxel wide background component, as shown on the right, but this tends to be cumbersome.

to different local regions of some larger segmentation. The size of the bounding box determines the *scale* of the local representation. This raises the question of how connectivity should be defined within these local regions. Two voxels may be connected only by a long path well outside the descriptor bounding box, as shown in fig. 3.7. As we would like the shape descriptors to be consistent with the local topology, such pairs should be considered disconnected.

To reliably distinguish between local and global connectivity, we represent segmentations globally as an *undirected graph over voxels.* The vertices of this graph correspond to positions in $\mathbb{Z}^3$, and edges are typically limited to occur between *neighboring* voxel positions, for some definition of neighboring.

Definition 7.3. The *von Neumann neighborhood* of $x \in \mathbb{Z}^3$ is the set $\{x' \mid \|x - x'\|_1 = 1\}$ of points at a Manhattan distance of 1 from x.

Remark. The von Neumann neighborhood is also called the 6-connectivity neighborhood in 3-D because it has cardinality 6.

Definition 7.4. The *Moore neighborhood* (or 26-connectivity neighborhood) of $x \in \mathbb{Z}^3$ is the set $\{x' \mid \|x - x'\|_\infty = 1\}$ of points at a L_∞ distance of 1 from x.

We will define our neighborhood $\mathcal{N}(x)$ to be the von Neumann neighborhood (6-connectivity), though the Moore neighborhood or any other (symmetric) neighborhood could equally well be used.

The segments themselves are *implicitly* defined by the connected components of this graph, in contrast to a representation defined by an explicit labeling of voxels by the component to which they belong. The advantage of this representation is illustrated in fig. 7.4.

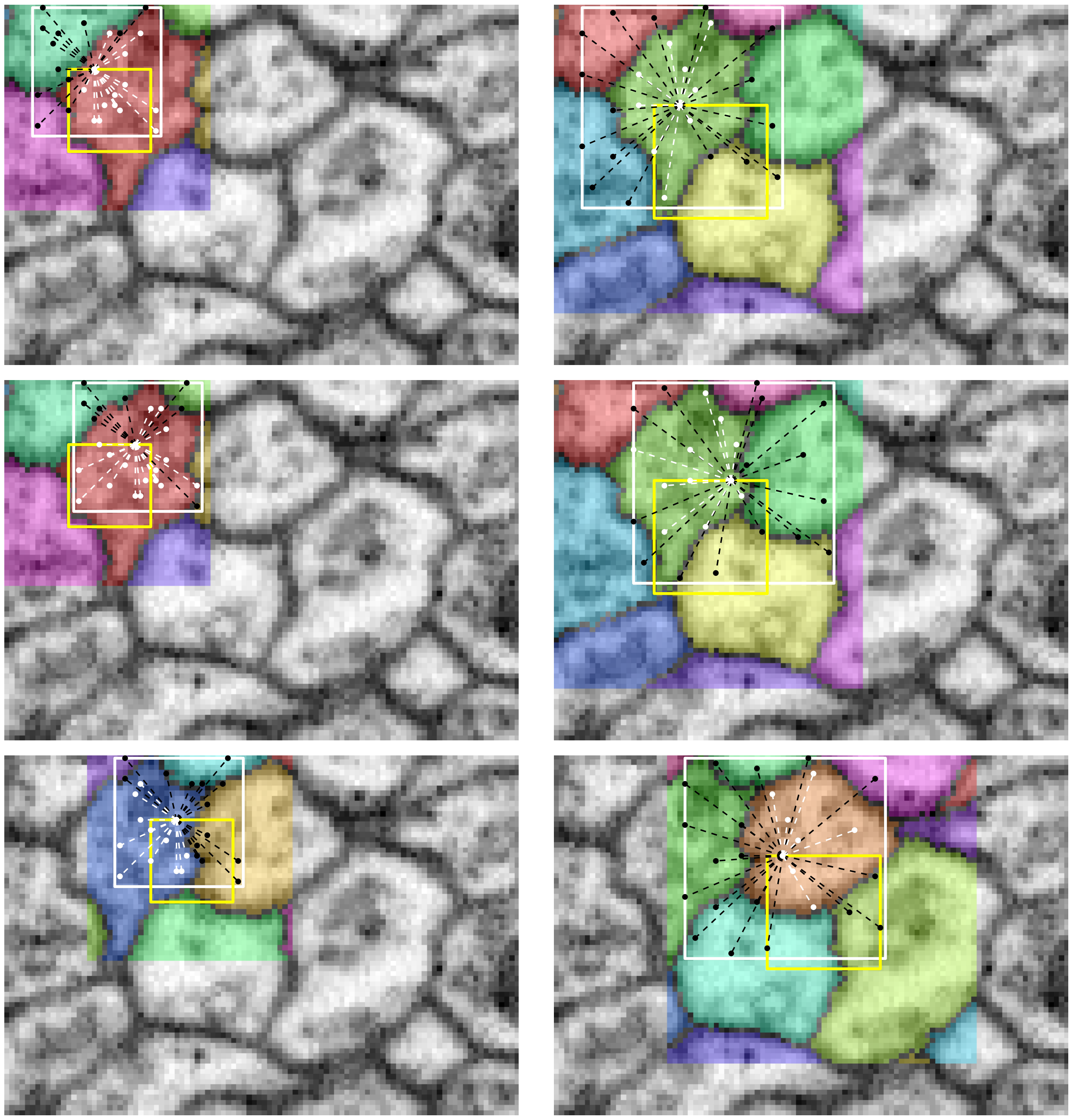
In order for shape descriptors to represent local connectivity within some larger segmentation S, we will compute each descriptor with respect to connectivity within some *connectivity region* C, a rectangular sub-region of the full bounds of S which necessarily contains one or more shape descriptor bounding boxes but may in general be significantly smaller than the full segmentation S; conceptually, the shape descriptor bounding box slides around to all possible positions contained within the connectivity region. (This sliding necessarily results in some minor inconsistency in context between different positions, but reduces computational and memory costs.) To obtain shape descriptors at all positions in S, we simply tile the space with overlapping rectangular connectivity regions of appropriate uniform size and stride, as shown in fig. 7.5. The connectivity region size determines the degree of locality of the connectivity information captured by the shape descriptor (independent of the descriptor bounding box size). It also affects computational costs, as described in section 7.5.

Definition 7.5. Given a shape descriptor specification s and connectivity region C, we denote by X_C^s the set of (type s) shape descriptor center positions for which the descriptor bounding box is contained within C.

Remark. Note that X_C^s is a rectangular region obtained by simply shrinking the rectangular region C by $(B_s - 1)/2$ on all sides.

We may wish to represent shape information at multiple scales, and to represent both the joint shape of nearby objects as well as the shape of individual objects. Therefore, rather than using a *single* shape descriptor specification s and a single connectivity region tiling, we use a *set* of shape descriptor specifications s, each implicitly associated with a particular choice of connectivity region size $\bar{B}_s$ and stride stride_s (specified by 3-D vectors of integers) that define a overlapped tiling of the full segmentation space.

Definition 7.6. Let $\mathcal{C}_s$ be the set of connectivity regions obtained as regular overlapping tiles of size $\bar{B}_s$ and stride stride_s.



(a) Small-scale (b) Large-scale

Figure 7.5: Connectivity region tiling. The connected components of the segmentation within each connectivity region C (shown in distinct colors) are maintained independently. The yellow rectangle within each connectivity region indicates the bounds of X_C^s, the set of (type s) shape descriptor center positions computed using C, which is simply the set of center positions for which the shape descriptor bounding box is contained within C. The white rectangle (of size B_s) indicates the bounding box of the shape descriptor (necessarily contained within C).

If we did not restrict the choice of stride$_s$, the bounding box for a shape descriptor at a given position might be fully contained within zero, one, or multiple connectivity regions, each of which would potentially result in a different binary descriptor. To avoid this complexity, we constrain $\bar{B}_s$ and stride$_s$ as follows:

$$\begin{aligned}&B_s \leq \bar{B}_s;\\&B_s = \bar{B}_s - \mathrm{stride}_s + 1.\end{aligned}$$

These constraints ensure that $\mathcal{C}_s$ *exactly partitions* the set of shape descriptor center positions, which allows us to make the following definition:

Definition 7.7. We denote by $\mathcal{C}_s(x)$ the single $C \in \mathcal{C}_s$ such that $x \in X^s_C$.

For convenience, we will also introduce some notation that applies to general undirected graphs that is relevant to our discussion:

Definition 7.8 (Connected components)**.** Given an undirected graph G, we denote by $\mathcal{K}(G)$ the partition of the vertex set of G into connected components, and denote by $K(v; G)$ the connected component G containing the vertex v.

Definition 7.9 (Induced subgraph)**.** Given an undirected graph G and a subset V' of its vertices, we denote by $G[V']$ the subgraph of G induced by V'.

While globally we will represent a segmentation S as a voxel graph, within a given connectivity region C we are concerned only with the connected components $\mathcal{K}(S[C])$ in the subgraph of S induced by C. Note that because the vertices of S correspond to voxels, i.e. positions in $\mathbb{Z}^3$, $\mathcal{K}(S[C]) \subset 2^{\mathbb{Z}^3}$. Based on these definition, we can more precisely state how local shape descriptors are defined.

Definition 7.10. Given a full segmentation S, for each shape descriptor specification s, we define the $|s|$-bit local binary shape descriptor $r_s(x; S)$ at position x by

$$r_s^{\{a,b\}}(x; S) := \mathbb{1}[K(x + a; S[C]) = K(x + b; S[C])] \qquad \text{for } \{a, b\} \in s,$$

where $C = \mathcal{C}_s(x)$.

Definition 7.11. Given a segmentation S, we define the *component visibility set* $V_s(x; S) \subseteq \mathcal{K}(S[C])$ of a position x to be the set of connected components at positions sampled by the shape descriptor s:

$$V_s(x; S) := \{K(x + c; S[C]) \,|\, c \in \{a, b\} \in s\},$$

where $C = \mathcal{C}_s(x)$.

Lemma 7.1. *Let a shape descriptor specification s, a position x, and segmentations S and S' be given. Let $C = \mathcal{C}_s(x)$. If $V_s(x; S) = V_s(x; S')$ (in particular if $\mathcal{K}(S[C]) = \mathcal{K}(S'[C])$), then $r_s(x; S) = r_s(x; S')$. Furthermore, in the case that s is center-based, then if $K(x; S[C]) = K(x; S'[C])$, then $r_s(x; S) = r_s(x; S')$.*

Proof. The first statement follows directly from definition 7.10.

To prove the second statement, suppose that s is center-based. Note that for all $\{a, b\} \in s$, $\{a, b\} = \{\vec{0}, c\}$ for $c \in \{a, b\}$. Thus, we have

$$\begin{aligned} r_s^{\{a,b\}}(x; S) &= r_s^{\{\vec{0},c\}}(x; S) \\ &= \mathbb{1}[K(x; S[C]) = K(x + c; S[C])] \\ &= \mathbb{1}[(x + c) \in K(x; S[C])] \qquad\qquad \text{for } \{\vec{0}, c\} = \{a, b\} \in s. \end{aligned}$$

The result follows. □

Remark. For general shape descriptor specifications s, $r_s(x; S)$ depends on S only by way of the subset of $\mathcal{K}(S[\mathcal{C}_s(x)])$ that are sampled, and for center-based shape descriptor specifications, $r_s(x; S)$ depends on S only by way of $K(x; S[\mathcal{C}_s(x)])$, the single component in $S[C]$ that contains x.

7.3 Conditional energy modeling of segmentations given images

Based on these binary shape descriptors, we define a global, translation-invariant energy model for predicting the cost of a complete segmentation S given a corresponding image I. This cost can be seen as analogous to the negative log-likelihood of the segmentation given the image, but we do not actually treat it probabilistically. Our goal is to define a model such that the true segmentation corresponding to a given image can be found by minimizing the cost; the energy can reflect both a prior over shape descriptors alone, as well as compatibility between shape descriptors and the image.

Definition 7.12. We denote by $\hat{E}_s(r; v)$ the local energy term that predicts the cost of a single shape descriptor r of type s given a corresponding image feature vector v.

As shown in fig. 7.7, we then define the global energy as

$$\begin{aligned} E_s(x; S; I) &\coloneqq \hat{E}_s\left(r_s(x; S); \phi(x; I)\right), \\ E(S; I) &\coloneqq \sum_s \sum_x E_s(x; S; I), \end{aligned}$$

where $\phi(x; I)$ denotes some feature representation of the image context centered around x.

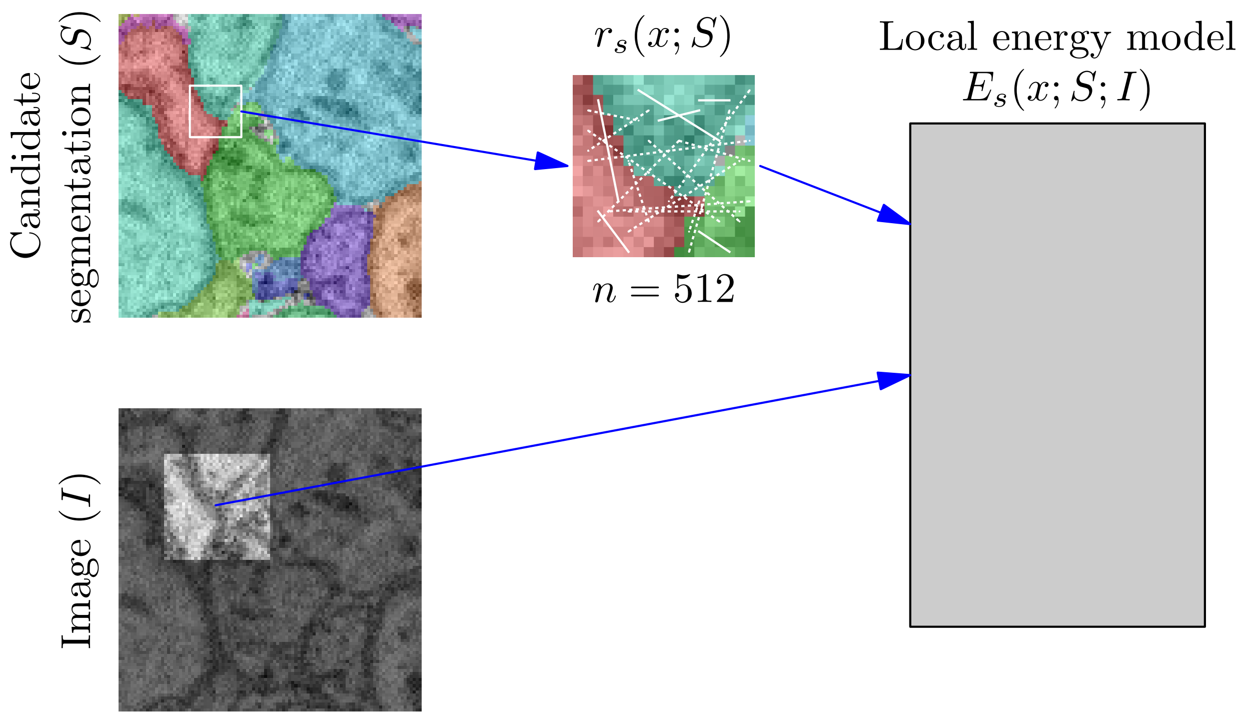


Figure 7.6: General architecture of local energy models.

Remark. It is a key property that the local energy terms may depend on any arbitrary representation of the local image context, but depend on the segmentation only by way of the binary shape descriptor representation.

To find (locally) minimal-cost segmentations under this model, we use local search over the space of agglomerations starting from some initial supervoxel segmentation. Using a simple greedy policy, at each step we consider all possible agglomeration actions, i.e. merges between any two adjacent supervoxels, and pick the action that results in the lowest energy. In terms of our undirected voxel graph representation, each merge action corresponds to a set of edges between adjacent voxels that will be added to the segmentation. By definition, agglomeration never results in the removal of voxel edges.

Definition 7.13. Given a segmentation S and a merge action e, we denote by $S + e$ the segmentation that results from merging e in S.

Remark. This binary $+$ operator over segmentations has the effect of taking the union of the edge sets, and is commutative.

To simplify the presentation, we will define the following notation for the forward discrete derivative:

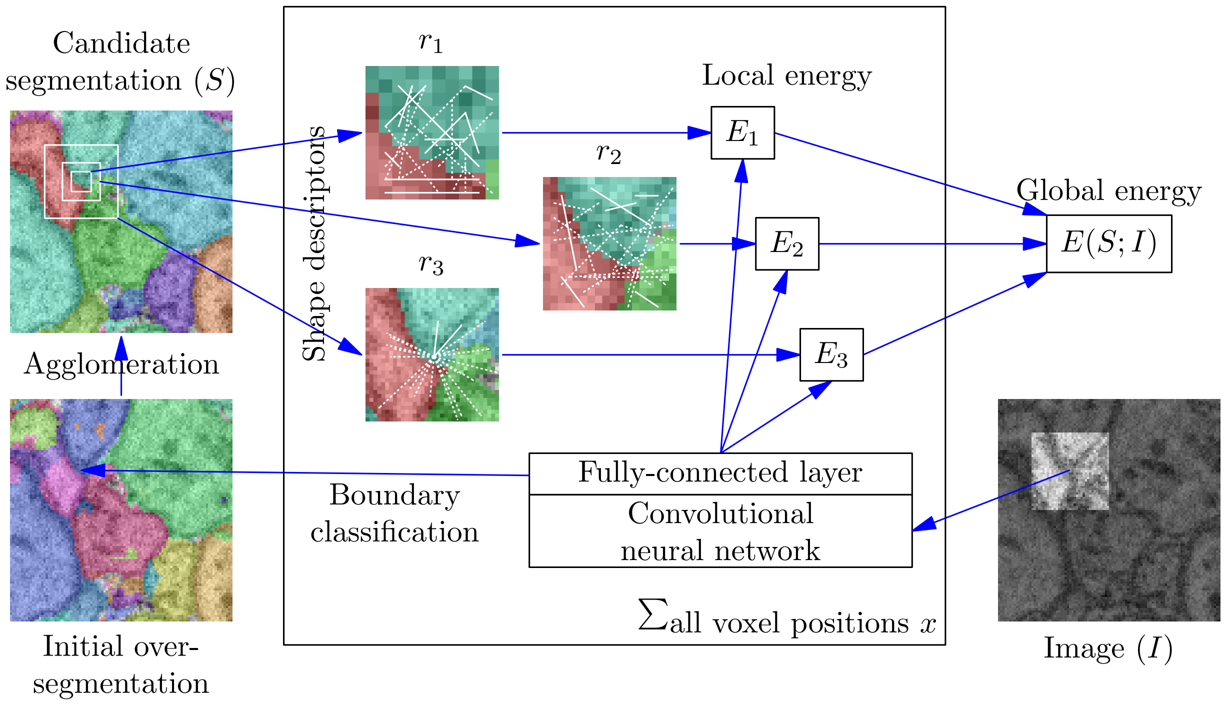


Figure 7.7: Illustration of computation of global energy for a *single* candidate segmentation S, using two pairwise and one center-based shape descriptor types. Connected and disconnected pairs within the shape descriptor are indicated by solid and dashed lines, respectively. The local energy $E_s(x; S; I) \in [0, 1]$, computed by a deep neural network, is summed over all shape descriptor types s and voxel positions x. Connectivity regions are not shown.

Definition 7.14 (Forward discrete derivative). We define $\Delta_S^{+e} f(S) \coloneqq f(S + e) - f(S)$ to be the forward discrete derivative of f with respect to S. Furthermore, we define

$$\Delta_S^{+e,+e'} f(S) \coloneqq \Delta_S^{+e} \Delta_S^{+e'} f(S)$$

to be the second-order discrete derivative.

Note that

$$\begin{aligned}\Delta_S^{+e,+e'} f(S) &= \Delta_S^{+e} \left[f(S + e') - f(S)\right] \\ &= \left[f(S + e' + e) - f(S + e)\right] - \left[f(S + e) - f(S)\right],\end{aligned}$$

and in the case that $+$ is commutative, as will always be the case in our use of this notation,

$$\Delta_S^{+e,+e'} f(S) = \Delta_S^{+e',+e}.$$

Based on this notation, we have the discrete derivative of the energy function

$$\Delta_S^{+e} E(S; I) = E(S + e; I) - E(S; I).$$

The greedy policy simply chooses at step t the action e that minimizes $\Delta_{S^t}^{+e} E(S^t; I)$, where S^t denotes the current segmentation at step t.

7.4 Energy model learning

We define the local energy model $\hat{E}_s\left(r; v\right)$ for each shape descriptor type/scale s by a learned neural network model that computes a real-valued score in $[0, 1]$ from a shape descriptor r and image feature vector v. Recall that the agglomeration decisions are based on the $\Delta_{S^t}^{+e} E(S^t; I)$ terms, which simply sum the energies from each position and descriptor type s. As in prior work [68], we treat this as a classification problem, with the goal of matching the sign of $\Delta_{S^t}^{+e} E(S^t; I)$ to $\Delta_{S^t}^{+e} \mathrm{error}(S^t, S^*)$, the corresponding change in segmentation error with respect to a ground truth segmentation S^*, measured using Variation of Information [65].

We tested two alternative training procedures: a simpler *local* training procedure that seeks to optimize the local $\hat{E}_s\left(r; v\right)$ scores independently, and a *global* training procedure that takes into account the contribution of these local scores to the overall $E(S; I)$ scores.

Local training procedure

We optimize the parameters of the energy model $\hat{E}_s\left(r; v\right)$ independently for each shape descriptor specification s. We seek to minimize the expectation

$$\mathbb{E}_i \Bigg[\ell(\Delta_{S_i}^{+e_i} \mathrm{error}(S_i, S^*), \hat{E}_s\left(r_s(x_i; S_i + e); \phi(x_i; I)\right)) + \tag{7.1}$$

$$\ell(-\Delta_{S_i}^{+e_i} \mathrm{error}(S_i, S^*), \hat{E}_s\left(r_s(x; S_i); \phi(x_i; I)\right)) \Bigg], \tag{7.2}$$

where i indexes over training examples that correspond to a particular sampled position x_i and a merge action e_i applied to a segmentation S_i. $\ell(y, a)$ denotes a binary classification loss function, where $a \in [0, 1]$ is the predicted probability that $y > 0$, weighted by $|y|$. Note that if $\Delta_{S_i}^{+e_i} \mathrm{error}(S_i, S^*) < 0$, then action e improved the score and therefore we want a low predicted score for the post-merge descriptor $r_s(x_i; S_i + e)$ and a high predicted score for the pre-merge descriptor $r_s(x_i; S_i)$; if $\Delta_{S_i}^{+e_i} \mathrm{error}(S_i, S^*) > 0$ the opposite applies. We tested the standard log loss $\ell(y, a) \coloneqq |y| \cdot [\mathbb{1}[y > 0] \log(a) + \mathbb{1}[y < 0] \log(1 - a)]$, as well as the signed linear loss $\ell(y, a) \coloneqq y \cdot a$, which more closely matches how the $E_s(x; S_i; I)$ terms contribute to the overall $\Delta_S^{+e} E(S; I)$ scores. We use stochastic gradient descent (SGD) to perform the optimization.

We obtain training examples by agglomerating using the *expert* policy that greedily optimizes $\mathrm{error}(S^t, S^*)$. At each segmentation state S^t during an agglomeration step (including the initial state), for each possible agglomeration action e, and each position x within the volume, we compute the shape descriptor pair $r_s(x; S^t)$ and $r_s(x; S^t + e)$ reflecting the pre-merge and post-merge states, respectively. If $r_s(x; S^t) \neq r_s(x; S^t + e)$, we emit a training example corresponding to this descriptor pair. We thereby obtain a conceptual stream of examples $\langle e, \Delta^{+e}_{S^t}\operatorname{error}(S^t, S^*), \phi(x; I), r_s(x; S^t), r_s(x; S^t + e)\rangle$.

This stream of examples may contain billions of examples (and many highly correlated), far more than required to learn the parameters of E_s. To reduce resource requirements, we use priority sampling [34], based on $|\Delta^{+e}_{S}\operatorname{error}(S, S^*)|$, to obtain a fixed number of weighted samples without replacement for each descriptor type s. We equalize the total weight of true merge examples ($\Delta^{+e}_{S}\operatorname{error}(S, S^*) < 0$) and false merge examples ($\Delta^{+e}_{S}\operatorname{error}(S, S^*) > 0$) in order to avoid learning degenerate models.[2]

Global Training

For the global training, we jointly optimize the parameters of the energy models E_s in order to minimize the expectation

$$\mathbb{E}_i\left[\ell(\Delta^{+e_i}_{S_i}\operatorname{error}(S_i, S^*), \operatorname{logistic}(\Delta^{+e_i}_{S_i}E(S_i; I)))\right].$$

We are able to do this with the same local training procedure through the use of a reweighting scheme: instead of applying SGD to the local objective with a fixed set of examples, we apply it to the reweighted objective (simultaneously for all shape descriptor types/scales s)

$$\mathbb{E}_i\Bigg[\ell(|g_i| \cdot \Delta^{+e_i}_{S_i}\operatorname{error}(S_i, S^*), \hat{E}_s(r_s(x_i; S_i + e); \phi(x_i; I)))+$$
$$\ell(-|g_i| \cdot \Delta_{e_i}\operatorname{error}(S_i, S^*), \hat{E}_s(r_s(x; S_i); \phi(x_i; I)))\Bigg],$$

where g_i is the gradient of the global loss with respect to $\Delta^{+e_i}_{S_i}E(S_i; I)$. We use a continuously resampled set of examples based on the current model parameters, obtained by priority sampling weighted by $|g_i \cdot \Delta^{+e_i}_{S_i}\operatorname{error}(S_i, S^*)|$.

7.5 Efficient energy minimization

Naïvely, computing the energy for just a single segmentation requires computing shape descriptors and then evaluating the energy model at every voxel position with the volume; a small volume may have tens or hundreds of millions of voxels. At each stage of the agglomeration, there may be thousands, or tens of thousands, of potential next agglomeration steps, each of which results in a unique segmentation. In order to choose the best next step, we must know the energy of all of these potential next segmentations. The computational cost to perform these computations *directly* would be tremendous.

We will discuss several computational tricks that allow us to efficiently compute these energy terms *incrementally*. Because the cost of evaluating the local energy model for a

[2]For example, if most of the weight is on false merge examples, as would often occur without balancing, the model can simply learn to assign a score that increases with the number of 1 bits in the shape descriptor.

single shape descriptor is many times more expensive than computing the shape descriptor, we structure our computation such that we only recompute a local energy term if the shape descriptor on which it depends has changed. This ensures that the total cost of evaluating the local energy terms is minimized, but even computing just the shape descriptors at each position within the volume for each potential agglomeration action at each step would still be prohibitively expensive. We therefore rely on geometric and region graph information to prune out the vast majority of this computation as well. Collectively, these tricks reduce the computational cost by several orders of magnitude; the effectiveness of these techniques is ultimately data-dependent, however.

Action representation

Recall that each agglomeration action e corresponds to a set of additional voxel edges to be added to the current segmentation state S^t. While in principle agglomeration could be defined with respect to arbitrary sets of voxel edges, we will carefully choose the set of actions to be considered in order to preserve the distinction between local and global connectivity while also allowing for a computationally-efficient implementation.

We will define actions in terms of adjacent supervoxels $K, K' \in \mathcal{K}(S^0)$ in the *initial* segmentation:

Definition 7.15. For any two distinct connected components $K, K' \in \mathcal{K}(S^0)$, let

$$e_{K,K'} \coloneqq \{\{x, x'\} \mid x' \in \mathcal{N}(x) \wedge (x, x') \in K \times K'\}.$$

Remark. If K and K' are not adjacent, then $e_{K,K'} = \varnothing$.

Note that we represent edges in the undirected voxel graph simply as two-element sets of voxel positions.

Definition 7.16. We define the supervoxel merge action set

$$A_S \coloneqq \{e_{K,K'} \neq \varnothing \mid K, K' \in \mathcal{K}(S) \wedge K \neq K'\}.$$

We will use $A^0 \coloneqq A_{S_0}$ as our set of actions for agglomeration. Note that each action corresponds to a *set* of voxel graph edges. At each step t of agglomeration, we choose an action $e^t \in A^t$. The set of remaining actions A^t after step t is simply the subset of actions in A that have not yet been performed, i.e. $A^{t+1} = A^t - \{e^t\}$. The segmentation state $S^{t+1} \coloneqq S^t + e^t$.

Definition 7.17. If e is a set of edges and C is a set of vertices, we denote by $e[C]$ the restriction of e to vertices in C, i.e. the subset of edges in e that are incident to two vertices in C. If S is a graph, we define $e[S] \coloneqq e[\text{vertices}(S)]$ to be the restriction of e to vertices in S.

Definition 7.18. Given a graph S and a partition T of vertices(S), we denote by G/T the contraction of G by T, i.e. vertices$(G/T) = T$ and

$$\text{edges}(G/T) = \{\{t_1, t_2\} \subseteq T \mid \exists e \in \text{edges}\, G : e \cap t_1 \neq \varnothing \wedge e \cap t_2 \neq \varnothing\}.$$

Definition 7.19. A set e of voxel edges is said to be a *supervoxel merge* in a voxel graph S of components $K, K' \in \mathcal{K}(S)$ if $e[S]$ is a non-empty set of edges between components K and K', or equivalently, that every edge in $e[S]$ corresponds to the edge $\{K, K'\}$ in $S/\mathcal{K}(S)$.

Definition 7.20. A set e of voxel edges is said to be a *redundant merge* in a voxel graph S, corresponding to the component $K \in \mathcal{K}(S)$, if $e[S]$ is a non-empty set of edges within component K, i.e. $\{\{K(a; S), K(b; S)\} \mid \{a, b\} \in e\} = \{\{K\}\}$, or equivalently, that every edge in e corresponds to a self edge $\{K\}$ in $S/\mathcal{K}(S)$.

Lemma 7.2. *If e is a redundant merge in S, then $\mathcal{K}(S + e) = \mathcal{K}(S)$.*

Proof. This follows from the fact that adding an edge between two vertices already part of the same connected component does not change set of connected components. □

Definition 7.21. Let e, e' be supervoxel merges in S. We say that e is *incident to a connected component* $K \in \mathcal{K}(S)$ in S if every edge in e is incident to a voxel in K, i.e. e is incident to K in $S/\mathcal{K}(S)$. We say that e is incident to e' in S if there exists $K \in componentsS$ to which both e and e' are incident, i.e. e is incident to e' in $S/\mathcal{K}(S)$.

Lemma 7.3. *If e is a supervoxel merge in S and S is a spanning subgraph of S', then e is a supervoxel merge or redundant merge in S'. If e is a redundant merge in S, then e is a redundant merge in S'.*

Proof. Suppose e is a supervoxel merge in S, corresponding to $K, K' \in \mathcal{K}(S)$. There must exist a components $J, J' \in \mathcal{K}(S')$ with $K \subseteq J$ and $K' \subseteq J'$. If $J = J'$, then e is a redundant merge in S'; otherwise e is a supervoxel merge of $\{J, J'\}$.

Suppose e is a redundant merge in S corresponding to $K \in \mathcal{K}(S)$. There must exist a component $J \in \mathcal{K}(S')$ with $K \subseteq J$. Hence, e is a redundant merge in S' corresponding to J. □

Remark. S is necessarily a spanning subgraph of $S + e$ for any merge action e.

Lemma 7.4. *At all steps t, all $e \in A$ are either supervoxel merges or redundant merges in S^t.*

Proof. This follows from lemma 7.3 and the fact that all $e \in A$ are supervoxel merges in S^0. □

The consequence of this lemma is that globally each merge action corresponds to a pair of connected components. Within the induced subgraph $S^t[C]$ of S^t restricted to a given connectivity region C, however, this lemma does not necessarily hold, even for $S^0[C]$, because a connected component of S^0 may correspond to more than one connected component of $S^0[C]$. For computational reasons that will be made apparent in section 7.5, we would like to ensure that it *does* hold, so that each merge action also corresponds to a pair of connected components within each connectivity region C (or is redundant within C).

To do this, we will assume that each connected component of S^0 is a clique. Our assumption sacrifices any distinction between local and global connectivity within the original supervoxels of S^0, but this is a small sacrifice given that they are expected to be small.

Lemma 7.5. *Given a connectivity region C, if e is a supervoxel merge in S^0 and $e[C]$ is non-empty, then e is either a supervoxel merge or a redundant merge in $S^t[C]$ for all t.*

Proof. Suppose e is a supervoxel merge in S^0 of components $K_1, K_2 \in \mathcal{K}(S^0)$. For all $\{a, b\} \in e[C]$, without loss of generality we can assume $a \in K_1$ and $b \in K_2$. By our assumption that K_1 and K_2 are cliques in S^0, $K_1 \cap C, K_2 \cap C \in \mathcal{K}(S^0[C])$. By the definition of $e[C]$, we have $a \in K_1 \cap C$ and $b \in K_2 \cap C$. Hence, e is a supervoxel merge in $S^0[C]$. The result follows from lemma 7.3. □

Δ representation

To efficiently implement a local search over agglomerations, at each step t of agglomeration, for each possible next agglomeration action e, we maintain the discrete derivative $\Delta_{S^t}^{+e} E(S^t; I)$, where S^t denotes the current segmentation at step t. Although our energy model is defined without any reference to supervoxels or merges, we prove a number of key properties that enable us to very efficiently compute and update these discrete derivative terms.

To maintain $\Delta_{S^t}^{+e} E(S^t; I)$, conceptually we must initially compute $\Delta_{S^0}^{+e} E_s(x; S^0; I)$ for each position x and action e, and then at each subsequent step t, agglomeration action a^t is taken and we update

$$\Delta_{S^{t+1}}^{+e} E(S^{t+1}; I) = \Delta_{S^t}^{+e} E(S^t; I) + \sum_s \sum_x \Delta_{S^t}^{+e,+e^t} E_s(x; S^t; I) \qquad \text{for all } e \in A^{t+1}.$$

Theorem 7.1 (Descriptor-based pruning)**.** *Let a position x and image I be given. Let $\bar{r}(S') \coloneqq r_s(x; S')$ and $\bar{E}(S') \coloneqq E_s(x; S'; I)$. Given a segmentation S, and merge e, if $\bar{r}(S) = \bar{r}(S + e)$, then $\Delta_S^{+e} \bar{E}(S) = 0$. Furthermore, for any merge e',*

$$\begin{aligned} d \coloneqq \Delta_S^{+e,+e'} \bar{E}(S) = \\ &+\bar{E}(S) \quad &-\bar{E}(S+e) \\ &-\bar{E}(S+e') \quad &+\bar{E}(S+e'+e), \end{aligned}$$

where some or all of the 4 terms can be canceled based on whether $\bar{r}(S) = \bar{r}(S+e)$, $\bar{r}(S) = \bar{r}(S+e')$, $\bar{r}(S+e) = \bar{r}(S+e'+e)$, and/or $\bar{r}(S+e') = \bar{r}(S+e'+e)$. In particular,

$$\begin{aligned}
\bar{r}(S) = \bar{r}(S+e) \ \wedge\ \bar{r}(S+e') = \bar{r}(S+e'+e) &\implies d = 0;\\
\bar{r}(S) \neq \bar{r}(S+e) \ \wedge\ \bar{r}(S+e') = \bar{r}(S+e'+e) &\implies d = \bar{E}(S) - \bar{E}(S+e);\\
\bar{r}(S) = \bar{r}(S+e) \ \wedge\ \bar{r}(S+e') \neq \bar{r}(S+e'+e) &\implies d = \bar{E}(S+e'+e) - \bar{E}(S+e').
\end{aligned}$$

By symmetry of the theorem with respect to e and e' we also have:

$$\begin{aligned}
\bar{r}(S) = \bar{r}(S+e') \ \wedge\ \bar{r}(S+e) = \bar{r}(S+e'+e) &\implies d = 0;\\
\bar{r}(S) = \bar{r}(S+e') \ \wedge\ \bar{r}(S+e) \neq \bar{r}(S+e'+e) &\implies d = \bar{E}(S+e'+e) - \bar{E}(S+e);\\
\bar{r}(S) \neq \bar{r}(S+e') \ \wedge\ \bar{r}(S+e) = \bar{r}(S+e'+e) &\implies d = \bar{E}(S) - \bar{E}(S+e').
\end{aligned}$$

Proof. For the first statement, if $\bar{r}(S) = \bar{r}(S+e)$, we have

$$\begin{aligned}
\bar{E}(S) &= \hat{E}_s\left(\bar{r}(S); \phi(x; I)\right)\\
&= \hat{E}_s\left(\bar{r}(S+e); \phi(x; I)\right)\\
&= \bar{E}(S+e).
\end{aligned}$$

The result follows.

The second statement is a straightforward result of the same cancellation principle. □

Remark. This theorem allows us to skip a large fraction of evaluations of the local energy model, which is in general significantly more expensive than just computing the shape descriptors (which must still be done in order to check the conditions of this theorem). If a packed bitvector representation is used, the cost of the descriptor comparisons is negligible.

Connectivity region-based pruning

Recall that for every merge action e exactly one of the following is true:

1. e is a supervoxel merge in $S^t[C]$;
2. e is a redundant merge in $S^t[C]$;
3. $e[C] = \varnothing$.

Definition 7.22. For each connectivity region C, we define the *active action set* $A^t[C] \subseteq A^t$ to be the subset of actions at step t that are supervoxel merges in $S^t[C]$.

Lemma 7.6. *Given a connectivity region C, if $e \notin A^t[C]$, then $e \notin A^{t'}[C]$ for all $t' > t$.*

Proof. Suppose $e \notin A^t[C]$. Then either $e[C] = \varnothing$ or e is a redundant merge in $S^t[C]$. If $e[C] = \varnothing$, then $e \notin A^{t'}[C]$ for any t'. Alternatively, if e is a redundant merge in $S^t[C]$, then since $S^t[C]$ is a spanning subgraph of $S^{t'}[C]$, by lemma 7.3 e is a redundant merge in $S^{t'}[C]$. □

Theorem 7.2 (Connectivity region-based pruning)**.** *Given a position x, time step t, and merge $e \in A^t$, let $C = \mathcal{C}_s(x)$ and let $A' = A^t[C]$. If $e \notin A'$, then $\Delta_{S^t}^{+e} E_s(x; S^t; I) = 0$. Furthermore, if $\{e, e'\} \not\subseteq A'$, then $\Delta_{S^t}^{+e,+e'} E_s(x; S^t; I) = 0$.*

Proof. We will begin by proving the first statement. Suppose $e \notin A'$. By definition of A', it follows that e is a redundant edge in $S^t[C]$, i.e. $\mathcal{K}(S^t[C]) = \mathcal{K}((S^t + e)[C])$. By lemma 7.1, we have $r_s(x; S^t) = r_s(x; S^t + e) = r$. The result follows from the first part of theorem 7.1.

Next we will consider the second statement. Since $\Delta_{S^t}^{+e,+e'} E_s(x; S^t; I) = \Delta_{S^t}^{+e',+e} E_s(x; S^t; I)$, the second statement is symmetric with respect to e and e'. It is sufficient, therefore to again consider the case that $e \notin A'$. By the first statement, $\Delta_{S^t}^{+e} E_s(x; S^t; I) = 0$. Since S^t is a spanning subgraph of $S^t + e'$, it is likewise the case that e is a redundant merge in $(S^t + e')[C]$, which implies that $r_s(x; S^t + e') = r_s(x; S^t + e' + e)$. The result follows from the second part of theorem 7.1. □

Remark. Because each action is typically active in only a tiny fraction of the connectivity regions, this theorem allows us to dramatically limit our computation.

Graph-based pruning

Lemma 7.7. *Let a segmentation S and a supervoxel merge e in S be given. Let $K \in \mathcal{K}(S)$ be a connected component of S. If e is not incident in S to K, then $K \in \mathcal{K}(S + e)$, i.e. merging e in S does not affect K.*

Proof. This follows from the fact that by definition of incidence of a supervoxel merge, no edge in e is incident to any voxel in K. □

Theorem 7.3 (Graph-based pruning)**.** *Suppose s defines a center-based descriptor. Let a segmentation S, position x, and supervoxel merges e and e' in S be given. Let $C = \mathcal{C}_s(x)$. If e is not incident in $S[C]$ to $K(x; S[C])$, then $r_s(x; S) = r_s(x; S + e)$ and $\Delta_S^{+e} E_s(x; S; I) = 0$. Furthermore, if e is not incident in $(S + e')[C]$ to $K(x; (S + e')[C])$, or e' is not incident to e in $S[C]$, then $\Delta_S^{+e,+e'} E_s(x; S; I) = 0$.*

Proof. We will being by proving the first statement. Suppose e is not incident in $S[C]$ to $K := K(x; S[C])$. By lemma 7.7, we have $K = K(x; (S + e)[C])$. By lemma 7.1 this implies that $r_s(x; S) = r_s(x; S + e)$. The result follows from the first part of theorem 7.1.

Next we will consider the second statement. Note that the condition that e is incident in $(S + e')[C]$ to $K(x; (S + e')[C])$ is equivalent to the condition that e is incident in $S[C]$ to $K := K(x; S[C])$, or e' is a supervoxel merge of K and K' in $S[C]$ (i.e. incident to e in $S[C]$) and e is incident to K' in $S[C]$.

There are two cases to consider: suppose e is not incident in $(S+e')[C]$ to $K(x;(S+e')[C])$. Then since S is a spanning subgraph of $S+e'$, it follows that e is also not incident in $S[C]$ to $K(x;S[C])$. The result follows from applying the first statement of the theorem to both S and $S+e'$ and then using theorem 7.1.

Alternatively, suppose e' is not incident to e in $S[C]$. This implies that $K(x;S[C])$ is incident to at most one of $\{e,e'\}$ in $S[C]$. By the symmetry of the theorem with respect to e and e', we will assume without loss of generality that e is not incident to $K(x;S[C])$ in $S[C]$. By our note above, we can infer that the condition for our first case, that e is not incident to $K(x;(S+e')[C])$ in $(S+e')[C]$, holds. □

Remark. This theorem demonstrates that for center-based descriptors, we can significantly limit computation based on the agglomeration graph structure. The cost of maintaining the incidence information is negligible.

Visibility-based pruning

Lemma 7.8. *Let a position x, segmentation S and supervoxel merge e of components K_1 and K_2 in $S[C]$, where $C = \mathcal{C}_s(x)$, be given. If e is incident in $S[C]$ to at most one component in $V_s(x;S)$, then $r_s(x;S) = r_s(x;S+e)$.*

Proof. There are two cases to consider. If e is not incident in $S[C]$ to any component in $V_s(x;S)$, then by lemma 7.7, $V_s(x;S) = V_s(x;S+e)$. The result follows from lemma 7.1. If e is incident in $S[C]$ to exactly one component $K_1 \in V_s(x;S)$, then $V_s(x;S+e') = V_s(x;S) + \{K_1' \cup K_2'\} - \{K_1\}$, i.e. merging e' in S adds additional voxels (not part of any visible component) to one visible component. Since these additional voxels are, by definition, not sampled by the shape descriptor, it follows that $r_s(x;S) = r_s(x;S+e)$. □

Theorem 7.4 (Visibility-based pruning)**.** *Given a position x, and segmentation S, let $C = \mathcal{C}_s(x)$ Let e' be a supervoxel merge of components K_1 and K_2 in $S[C]$. If e' is not incident in $S[C]$ to any component $K_1 \in V_s(x;S)$, then $\Delta_S^{+e} E_s(x;S;I) = 0$, and $\Delta_S^{+e,+e'} E_s(x;S;I) = 0$ for all supervoxel merges e in $S[C]$. If e' is incident in $S[C]$ to exactly one component $K_1 \in V_s(x;S)$, then for all supervoxel merges e of K_1', K_2' in $S[C]$ not incident to K_2 in $S[C]$, i.e. $K_2 \notin \{K_1', K_2'\}$, $\Delta_S^{+e,+e'} E_s(x;S;I) = 0$.*

Proof. To prove the first statement, suppose e' is not incident in $S[C]$ to any component in $V_s(x;S)$. For any supervoxel merge e in $S[C]$, it must be the case that e' is incident in $(S+e)[C]$ to at most one component in $V_s(x;S+e)$. By applying lemma 7.8 to both $S[C]$ and $S[C+e]$, we have $r_s(x;S) = r_s(x;S+e')$ and $r_s(x;S+e) = r_s(x;S+e+e')$. The result follows from theorem 7.1.

To prove the second statement, suppose e' is incident in $S[C]$ to exactly one component $K_1 \in V_s(x;S)$. As for the first statement, by lemma 7.8 we have $r_s(x;S) = r_s(x;S+e')$. Let e be a supervoxel merge of K_1', K_2' in $S[C]$ not incident to K_2 in $S[C]$. If e is incident to K_1, then e' is incident in $(S+e)[C]$ to exactly one component $(K_1' + K_2') \supseteq K_1$. If e

is not incident to K_1, then e' is incident in $(S+e)[C]$ to exactly the one component K_1. Therefore, by lemma 7.8 we have $r_s(x; S+e) = r_s(x; S+e+e')$ and the result follows from theorem 7.1. □

Determining whether a given component $K \in S[C]$ is a member of the *exact* visibility set $V_s(x; S)$ for all positions $x \in X_C^s$ is computationally expensive, i.e. $\Theta(|X_C^s| \cdot |s|)$. However, to satisfy the conditions of theorem 7.4, it is sufficient to check membership in any *superset* of the visibility set; this restricts the conditions under which pruning is done, but we can choose a superset in which membership can be checked much more efficiently.

Definition 7.23. For d-dimensional vectors $\vec{a}, \vec{b} \in \mathbb{Z}^d$, we denote by $R_{\vec{a}}^{\vec{b}}$ the hyperrectangle

$$R_{\vec{a}}^{\vec{b}} \coloneqq \{\vec{x} \in \mathbb{Z}^d \mid \vec{a} \le \vec{x} < \vec{b}\}.$$

Definition 7.24. Given a segmentation S, we define the *approximate component visibility set* $\hat{V}_s(x; S) \subseteq \mathcal{K}(S[C])$ of a position x to be the set of connected components at positions within a bounding box of size B_s centered at x:

$$\hat{V}_s(x; S) \coloneqq \left\{ K(x+c; S[C]) \,\middle|\, c \in R_{-(B_s-\vec{1})/2}^{(B_s-\vec{1})/2} \right\},$$

where $C = \mathcal{C}_s(x)$.

Lemma 7.9. *Given a segmentation S, $\hat{V}_s(x; S) \subseteq V_s(x; S)$.*

Proof. This follows from the fact that $\{a, b\} \subset R_{-(B_s-\vec{1})/2}^{(B_s-\vec{1})/2}$ for all $\{a, b\} \in s$. □

Definition 7.25. For two coordinate vectors a and b, $a \odot b$ denotes the element-wise product.

For a given component $K \in S[C]$, by first computing a summed area table [29], we can efficiently determine whether $K \in \hat{V}_s(x; S[C])$ for all positions $x \in X_C^s$, as described in algorithm 7.1. The computational cost is $\Theta(|C|)$. To check the conditions of theorem 7.4 for a given supervoxel merge e' of $K_1, K_2 \in S[C]$, we simply apply algorithm 7.1 to both K_1 and K_2. Alternatively, to check only the (more limited) first condition that $\{K_1, K_2\} \cap V_s(x; S) = \varnothing$, then it is sufficient to apply algorithm 7.1 just once to $K_1 \cup K_2$.

At agglomeration steps $t > 0$, we can apply theorem 7.4 with $e' = a^{t-1}$ and $e \in A^t[C]$ in order to limit the set of positions x and edges e for which the change in local energy $\Delta_{S^t}^{+e,+e^t} E_s(x; S^t; I)$ must be computed. In principle, we could apply theorem 7.4 to all candidate actions $e' \in A^t[C]$ at a given agglomeration step t, but this would require computing separate summed area tables for all components $K \in \mathcal{K}(S^t[C])$ incident to a candidate action, which would involve considerable overhead. Therefore in practice the theorem is only applicable for $t > 0$.

Algorithm 7.1 Optimized membership test for approximate component visibility sets.

```
Require: $(G, +)$ is a commutative group with identity $0_G$.
 1: function ComputeSummedAreaTable($A\colon R_a^b \to G,\ R_a^b$)
 2:     Declare array $T\colon R_a^{b+\vec{1}} \to G$
 3:     for $x \in R_a^{b+\vec{1}} : \|x - a\|_0 < d$ do
 4:         $T(x) \leftarrow 0_G$
 5:     end for
 6:     for $x \in R_{a+\vec{1}}^{b}$ do    ▷ Iteration over $x$ must respect the usual partial ordering on $\mathbb{Z}^d$.
 7:         $T(x) \leftarrow A(x - \vec{1}) + \sum_{z \in \{0,1\}^d - \{\vec{0}\}} (-1)^{1+\|z\|_1} \cdot T(x - z)$
 8:     end for
 9:     return $T$
10: end function
11: function SummedAreaTableLookup($T\colon R_a^{b+\vec{1}} \to G,\ R_{a'}^{b'} \subseteq R_a^b$)
12:     return $\sum_{z \in \{0,1\}^d} (-1)^{\|z\|_1} \cdot T(b + (a - b) \odot z)$
13: end function
14: function ComputePositionsWithVisibility($s,\ S,\ C,\ K \in S[C]$)
15:     Define $A(x) \coloneqq \mathbb{1}[K = K(x; S[C])]$
16:     $T \leftarrow$ ComputeSummedAreaTable($A, C$)
17:     $X \leftarrow \varnothing$
18:     for $x \in X_C^s$ do
19:         if SummedAreaTableLookup($T, R_{x-(B_s-\vec{1})/2}^{x+(B_s-\vec{1})/2}$) $> 0$ then
20:             $X \leftarrow X \cup \{x\}$
21:         end if
22:     end for
23:     return $X$
24: end function
```

Zone-based pruning

In the case of a pairwise shape descriptor specification s, we cannot apply theorem 7.3, and consequently based only on theorem 7.2, for each position x we must compute shape descriptors for all actions $e \in A^t[\mathcal{C}_s(x)]$. Theorem 7.4 primarily allows us to prune positions x but not actions e, and is not applicable at the initial state $t = 0$.

At $t = 0$, the number of positions that must be considered within a given connectivity region C is exactly $|X^s_C|$; at later steps $t > 0$ the number of positions may be reduced due to theorem 7.4 but nonetheless tends to grow linearly with $|X^s_C|$. The size of the active set $A^t[C]$ tends to grow superlinearly in $|C|$. Hence, the computational cost of shape descriptor computation based only on the pruning theorems we've introduced thus far grows superquadratically in $|C|$.

To mitigate this effect, we could of course simply ensure that connectivity regions are very small. A larger number of small connectivity regions does, however, introduce additional overhead, as explained in section 7.6, and therefore may actually increase the computational cost. Furthermore, reducing the connectivity region size also affects the extent to which shape descriptors reflect local or global connectivity, and we would like to be able to choose that independently of computational concerns.

We therefore introduce a subdivision of connectivity regions into *zones*.

Definition 7.26. For each connectivity region $C \in \mathcal{C}_s$, the *zone set* $\mathcal{Z}_{s,C}$ is a partition of X^s_C.

We can extend our definition of component visibility sets, previously defined only for individual positions in definition 7.11, to sets of positions:

Definition 7.27. The *component visibility set* $W_s(Z; S)$ for a zone Z is defined by

$$W_s(Z; S) := \cup_{x \in Z} V_s(x; S).$$

Definition 7.28. The *zone visibility set* $W_s^{-1}(K; C)$ is the set of zones whose component visibility set contains K:

$$W_s^{-1}(K; C) := \{Z \in \mathcal{Z}_{s,C} \mid K \in W_s(Z; S)\},$$

where S is some segmentation for which $K \in \mathcal{K}(S[C])$.

Remark. The zone visibility set does not depend on the segmentation S beyond the fact that $K \in \mathcal{K}(S[C])$. By definition, a merge that does not affect a connected component K' does not affect its zone visibility set $W_s^{-1}(K'; C)$.

Theorem 7.5. *Given a supervoxel merge e of K_1, K_2 in $S[C]$, merging e in S has the effect of merging the zone visibility sets of K_1 and K_2:*

$$W_s^{-1}(K_1 \cup K_2; C) = W_s^{-1}(K_1; C) \cup W_s^{-1}(K_2; C).$$

Proof. To show that the $W_s^{-1}(K_1;C) \cup W_s^{-1}(K_2;C)$ contains $W_s^{-1}(K_1 \cup K_2;C)$, let $Z \in W_s^{-1}(K_1 \cup K_2;C)$ be given. Then $\exists x \in Z, c \in \{a,b\} \in s$ such that $K(x+c; S'[C]) = (K_1 \cup K_2)$, where S' is the segmentation that results from the merge of K_1 and K_2 in S. Hence, $K(x+c;S[C]) \subset \{K_1, K_2\}$, and it follows that $Z \in W_s^{-1}(K_1;C) \cup W_s^{-1}(K_2;C)$.

To show that $W_s^{-1}(K_1 \cup K_2;C)$ contains $W_s^{-1}(K_1;C) \cup W_s^{-1}(K_2;C)$, let $Z \in W_s^{-1}(K_1;C)$ be given. Then $\exists x \in Z, c \in \{a,b\} \in s$ such that $K(x+c;S[C]) = K_1$. It follows that $K(x+c;S'[C]) = (K_1 \cup K_2)$, and therefore $Z \in W_s^{-1}(K_1 \cup K_2;C)$. □

Definition 7.29. A supervoxel merge e of K_1, K_2 in $S[C]$ is said to be *active in zone* Z of $S[C]$ if $Z \in W_s^{-1}(K_1;C) \cap W_s^{-1}(K_2;C)$.

Definition 7.30. We denote by $A_Z^t[C]$ the *active action set of zone* Z of connectivity region C at time t, the set of actions e in $A^t[C]$ that are active in zone Z of $S^t[C]$.

Theorem 7.6. *If a supervoxel merge e in $S[C]$ is not active in zone Z, then for all positions $x \in Z$ we have $r_s(x;S) = r_s(x;S+e)$, $\Delta_S^{+e} E_s(x;S;I) = 0$. Furthermore, given a supervoxel merge e' in $S[C]$, if a supervoxel merge e in $(S+e')[C]$ is not active in zone Z, then for all positions $x \in Z$, $\Delta_S^{+e,+e'} E_s(x;S;I) = 0$.*

Proof. To prove the first statement, suppose the supervoxel merge e in $S[C]$ of components K, K' is not active in zone Z, and $x \in Z$. Then by definition 7.29, $\{K, K'\} \not\subseteq V_s(x;S)$. Hence, by lemma 7.8 $r_s(x;S) = r_s(x;S+e)$, and the result follows from theorem 7.1.

To prove the second statement, suppose the supervoxel merge e of components K_1, K_2 in $(S+e')[C]$ is not active in zone Z of $(S+e')[C]$. By the first statement of the theorem, this implies that $r_s(x;S+e') = r_s(x;S+e'+e)$ for all $x \in Z$. It remains to be shown that e is also not active in zone Z of $S[C]$. By definition 7.29,

$$Z \notin W_s^{-1}(K_1;C) \cap W_s^{-1}(K_2;C). \tag{7.3}$$

There are two cases to consider:

1. If e is not incident to e' in $S[C]$, then $\{K_1, K_2\} \subset \mathcal{K}(S[C])$ and it follows from eq. (7.3) and definition 7.29 that e is also not active in zone Z of $S[C]$.

2. Alternatively, if e is incident to e' in $S[C]$, then without loss of generality we can assume that $K_2 \subset \mathcal{K}(S[C])$ and $K_1 = K_1' \cup K_2'$, where e' is a supervoxel merge of K_1', K_2' in $S[C]$, and e is a supervoxel merge of K_1', K_2 in $S[C]$. Then by theorem 7.5,

$$\begin{aligned} W_s^{-1}(K_1;C) &= W_s^{-1}(K_1' \cup K_2';C) \\ &= W_s^{-1}(K_1';C) \cup W_s^{-1}(K_1';C). \end{aligned}$$

   It follows that $Z \notin W_s^{-1}(K_1';C) \cap W_s^{-1}(K_2;C)$, which by definition 7.29 implies that e is not active in zone Z of $S[C]$. By the first statement of the theorem, we have $r_s(x;S) = r_s(x;S+e)$ for all $x \in Z$. The result follows from theorem 7.1. □

If we ensure that the total number of zones, $|\mathcal{Z}_{s,C}|$, is limited to a small constant, e.g. 64, then we can efficiently represent the zone visibility set $W_s^{-1}(K; C)$ for each component K as a bit vector. Maintaining these visibility sets over the course of agglomeration, per theorem 7.5, requires only bitwise disjunction operations; determining whether a supervoxel merge e is active in a zone Z, per definition 7.29, requires only bitwise conjunction.

Based on theorem 7.6, the cost of computing all unpruned shape descriptors within a connectivity region C can be formulated as

$$\sum_{Z \in \mathcal{Z}_{s,C}} \left[(|Z| + \alpha) \cdot |A_Z^t[C]| \right] + \beta(|\mathcal{Z}_{s,C}|),$$

where α represents the overhead per action active in a zone, and β is a non-decreasing function that specifies an additional overhead for a given number of zones; α and β may represent either computational or memory costs.

Minimizing this cost exactly is in general a hard integer programming problem. We find a locally-optimal solution using an approach that mirrors our approach for minimizing the global energy $E(S; I)$: we start with an initial set of zones, either single-voxel zones or a regular grid, and greedily merging zones in order to reduce the cost.

7.6 Implementation

A high-performance implementation of our agglomeration procedure is critical for testing and applying it to the large datasets inherent to neuronal reconstruction. The implementation challenges are, however, considerable:

- Conceptually the local search over the space of agglomerations depends on the value of an enormous number of distinct local energy terms.
- The pruning tricks described in section 7.5 greatly reduce the number of shape descriptor and local energy model computations, but at the cost of significant algorithmic complexity.
- We wish to be able to use a high-dimensional image feature representation $\phi(x; I)$. Storing the precomputed 512-dimensional image features over just as a small 256^3 voxel volume in 32-bit floating point format requires 34 GB of memory. While in absolute terms this is not a large amount of memory, it limits the number of independent volumes that may be agglomerated in parallel on a single machine, and for reasonable cost-effectiveness it is necessary, therefore, that a single agglomeration be able to take advantage of multiple cores.
- The computational steps required are not primarily standard operations like convolutions, Fourier transforms, matrix multiplications, or other linear algebra operations for which there has already been extensive study of efficient implementation techniques

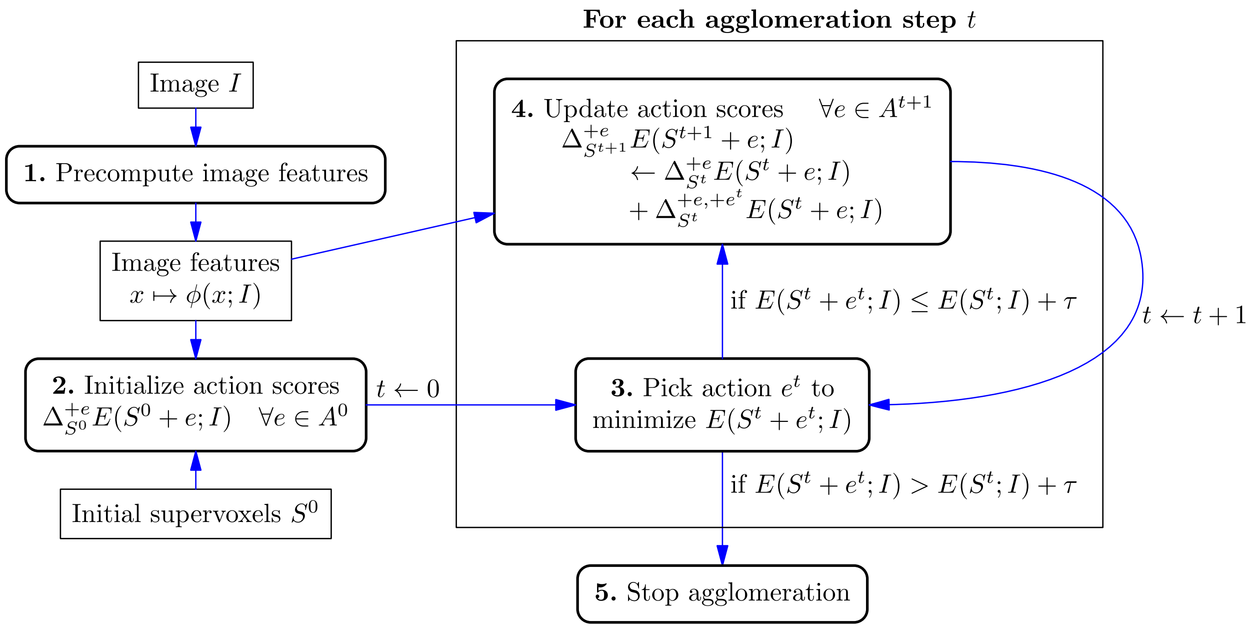

For each agglomeration step t
Image I
1. Precompute image features
Image features $x \mapsto \phi(x; I)$
2. Initialize action scores $\Delta^{+e}_{S^0} E(S^0 + e; I) \quad \forall e \in A^0$
Initial supervoxels S^0
$t \leftarrow 0$
4. Update action scores $\forall e \in A^{t+1}$ $\Delta^{+e}_{S^{t+1}} E(S^{t+1} + e; I) \leftarrow \Delta^{+e}_{S^t} E(S^t + e; I) + \Delta^{+e,+e^t}_{S^t} E(S^t + e; I)$
if $E(S^t + e^t; I) \leq E(S^t; I) + \tau$
$t \leftarrow t + 1$
3. Pick action e^t to minimize $E(S^t + e^t; I)$
if $E(S^t + e^t; I) > E(S^t; I) + \tau$
5. Stop agglomeration

Figure 7.8: High-level CELIS agglomeration procedure. Arrows show the flow of data (indicated by rectangles) and control (indicated by rounded rectangles). At a high-level, agglomeration proceeds in a sequential manner. At each agglomeration step t, the next action e^t if selected to greedily minimize the global energy $E(S^t + e^t; I)$. If the best e^t decreases the energy by more than τ, i.e. $\Delta^{+e^t}_{S^t} E(S^t; I) < \tau$, then agglomeration continues. Otherwise, agglomeration terminates. To save computation at the cost of greater memory use, the image feature vector $\phi(x; I)$ for all positions x are precomputed prior to the start of agglomeration. The parallel pipeline used to initialize and update the action scores (steps 2 and 4) is shown in detail in fig. 7.10; the details of the data structures that are updated by these steps are shown in fig. 7.9.

and for which high-performance implementations (for single and multiple CPU cores, as well as for GPU platforms) are already available.

To address these challenges, we designed a parallel pipeline that, at agglomeration step t, determines which shape descriptors and local energy terms need to be computed, performs those computations, and updates $\Delta^{+e}_{S^t} E(S^t; I)$ for candidate actions e, in order that the action e that greedily minimizes $E(S^t + e; I)$ may be chosen.

Data structures maintained during agglomeration

This pipeline is based around several interlinked data structures, as shown in fig. 7.9:

- The initial segmentation S^0 serves to define the agglomeration space over which our local search will operate. While conceptually we represent segmentations as an undi-

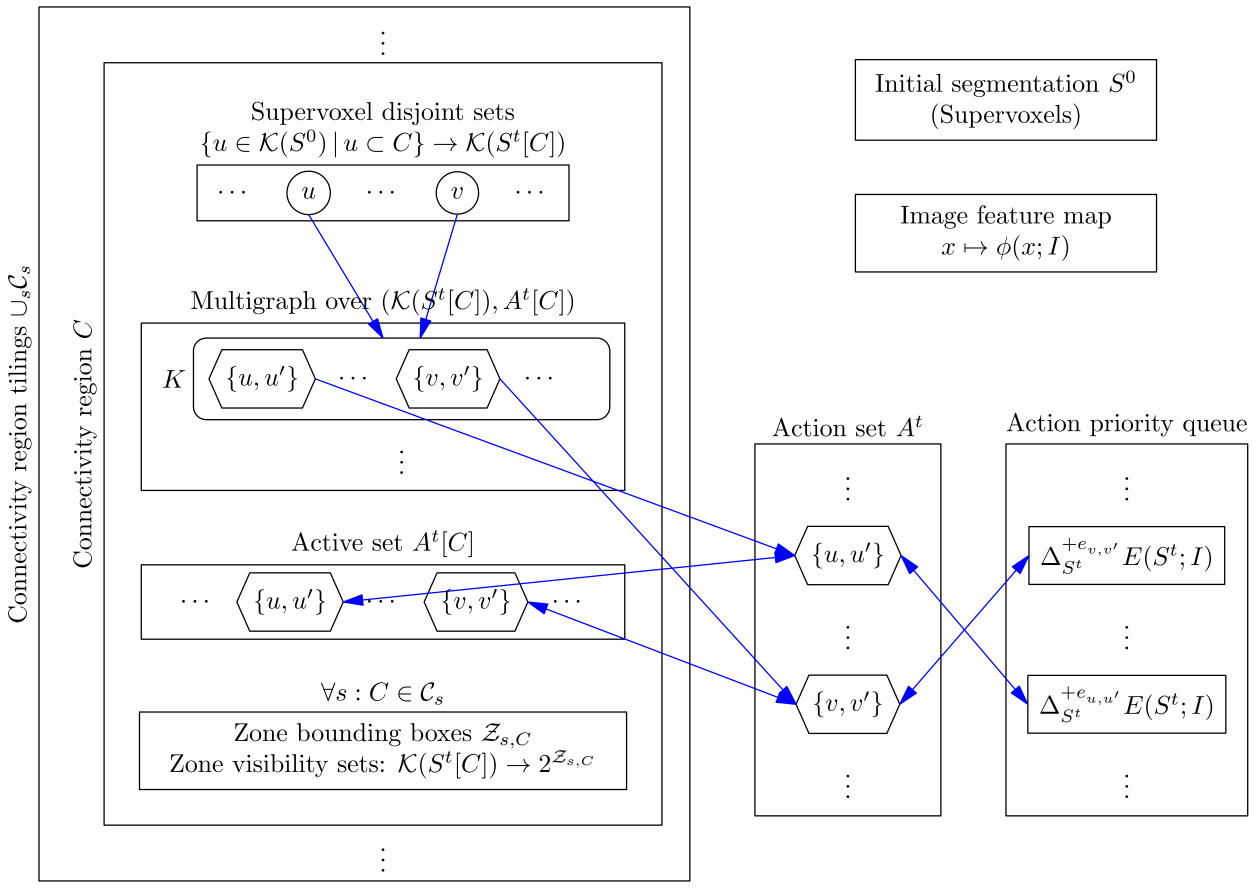

Connectivity region tilings $\cup_s \mathcal{C}_s$
Connectivity region C
Supervoxel disjoint sets
$\{u \in \mathcal{K}(S^0) \mid u \subset C\} \to \mathcal{K}(S^t[C])$
u
v
Multigraph over $(\mathcal{K}(S^t[C]), A^t[C])$
K
$\{u, u'\}$
$\{v, v'\}$
Active set $A^t[C]$
$\forall s : C \in \mathcal{C}_s$
Zone bounding boxes $\mathcal{Z}_{s,C}$
Zone visibility sets: $\mathcal{K}(S^t[C]) \to 2^{\mathcal{Z}_{s,C}}$
Initial segmentation S^0
(Supervoxels)
Image feature map
$x \mapsto \phi(x; I)$
Action set A^t
Action priority queue
$\Delta^{+e_{v,v'}}_{S^t} E(S^t; I)$
$\Delta^{+e_{u,u'}}_{S^t} E(S^t; I)$

Figure 7.9: Data structures for implementing CELIS agglomeration. Arrows indicate the links that make up the data structures.
For each connectivity region C, we maintain a disjoint sets data structure that maps supervoxels $u \in \mathcal{K}(S^0)$ to connected components of $K \in \mathcal{K}(S^t[C])$. For each connected component K, we maintain a list of incident supervoxel merge actions $e \in A^t$ to allow for efficient application of theorem 7.3. This represents the multigraph obtained by contracting the connected components of $S^t[C]$. For each shape descriptor specification s for the connectivity region is used, we also maintain the zone information and zone visibility sets (represented as bit vectors) for each connected component K.
For each action $e \in A^t$, we store $\Delta^{+e}_{S^t} E(S^t; I)$, which serves as the ordering key for a priority queue over actions used for greedy agglomeration. We also maintain for each action e the set of connectivity regions for which $e \in A^t[C]$, for efficient application of theorem 7.2.

rected graph over voxels (as described in section 7.2), we assume for simplicity that each initial supervoxel, i.e. connected component $u \in \mathcal{K}(S^0)$, is a clique, as described in section 7.5. This allows us to unambiguously represent S^0 by labeling each voxel with an integer that uniquely identifies the supervoxel that contains it. Note that it is *not* in general the case that the connected components of S^t at steps $t > 0$ are cliques, meaning that we cannot unambiguously represent S^t by a component labeling. In fact we do not explicitly represent the segmentation S^t at later steps; instead it is represented implicitly by the initial segmentation S^0 and the sequence of actions $a^1, \ldots, a^t$ that have been performed.

- The global set of actions A^t. As described in section 7.5, each action $e \in A^t$ corresponds to a pair $\{u, v\} \subset \mathcal{K}(S^0)$, i.e. $e = e_{u,v}$. We represent each action $e_{u,v}$ by the pair of integer identifiers corresponding to the supervoxels u and v. The action set at any step $t > 0$ is simply $A^0 - \{e^{t'} \mid t' < t\}$. For each action $e \in A^t$, we also maintain the set of connectivity regions C in which it is active, i.e. $e \in A^t[C]$. This allows theorem 7.2 to be applied efficiently. Recall that by lemma 7.6, actions are removed from $A^t[C]$ during the course of agglomeration, but are never added. Thus, once an action e is no longer active in any connectivity region, it ceases to affect the global energy.

- The key information that the pipeline serves to maintain is the change in global energy, $\Delta^{+e}_{S^t} E(S^t; I)$, that would result from merging each action e. This change in energy is essentially a *score* associated with the action. Our agglomeration procedure follows the greedy policy of choosing at each step the action with the lowest (i.e. most negative) score. Therefore, for each active set e we store the associated score, and we also maintain a priority queue over the scores, to allow for efficiently finding the edge with the lowest score.

- Another major component is a data structure representing the set of connectivity regions, i.e. the union of the connectivity region tilings $\mathcal{C}_s$ for each shape descriptor specification s.[3] The set of connectivity regions remains fixed throughout agglomeration. For each connectivity region, we maintain the following information:

  - A mapping from global supervoxels $u \in \mathcal{K}(S^0)$ (represented by unique integer identifiers) to connected components $K \in \mathcal{K}(S^t[C])$ within the connectivity region (also represented by unique integer identifiers within each connectivity region, separate from the global supervoxel identifiers). We handle the mapping of global

[3]In the typical case that different tile sizes $\bar{B}_s$ and strides stride$_s$ are used for each specification s, these tilings $\mathcal{C}_s$ will be disjoint (but certainly overlapping, as they cover the same space), meaning that each connectivity region C is associated with only one specification s. In general, though, there may be multiple shape descriptor specifications s for which C is used, i.e. $C \in \mathcal{C}_s$. Sharing a connectivity region for multiple shape descriptor specifications slightly reduces memory and computational overhead, because the per-connectivity region data structures, namely the connected components $\mathcal{K}(K)$ and active action sets $A^t[C]$, only have to be stored and maintained once.

supervoxel identifiers using a hash table, and we maintain the connected components using a standard disjoint sets data structure based on union by rank and path compression. [27, p. 505]

- The active set $A^t[C]$ of actions that affect connectivity within C.
- For each component $K \in \mathcal{K}(S^t[C])$, the set of incident actions $e \in A^t[C]$. Each incident action corresponds to a supervoxel merge of K and some other component $K' \in \mathcal{K}(S^t[C])$ in $S^t[C]$. There may, however, be two distinct actions $e, e' \in A^t[C]$ that are both supervoxel merges of the same two components K and K'. These sets of incident actions therefore correspond to the adjacency lists of the multigraph $S^t[C]/\mathcal{K}(S^t[C])$.
- For each shape descriptor specification s for which $C \in \mathcal{C}_s$ (typically there may only be one such s), we additionally maintain:
  * The partition $\mathcal{Z}_{s,C}$ of X^s_C. We represent each zone compactly as the union of disjoint rectangular regions.
  * For each component $K \in \mathcal{K}(S^t[C])$, the zone visibility set $W_s^{-1}(K; C)$ represented as a bit vector.

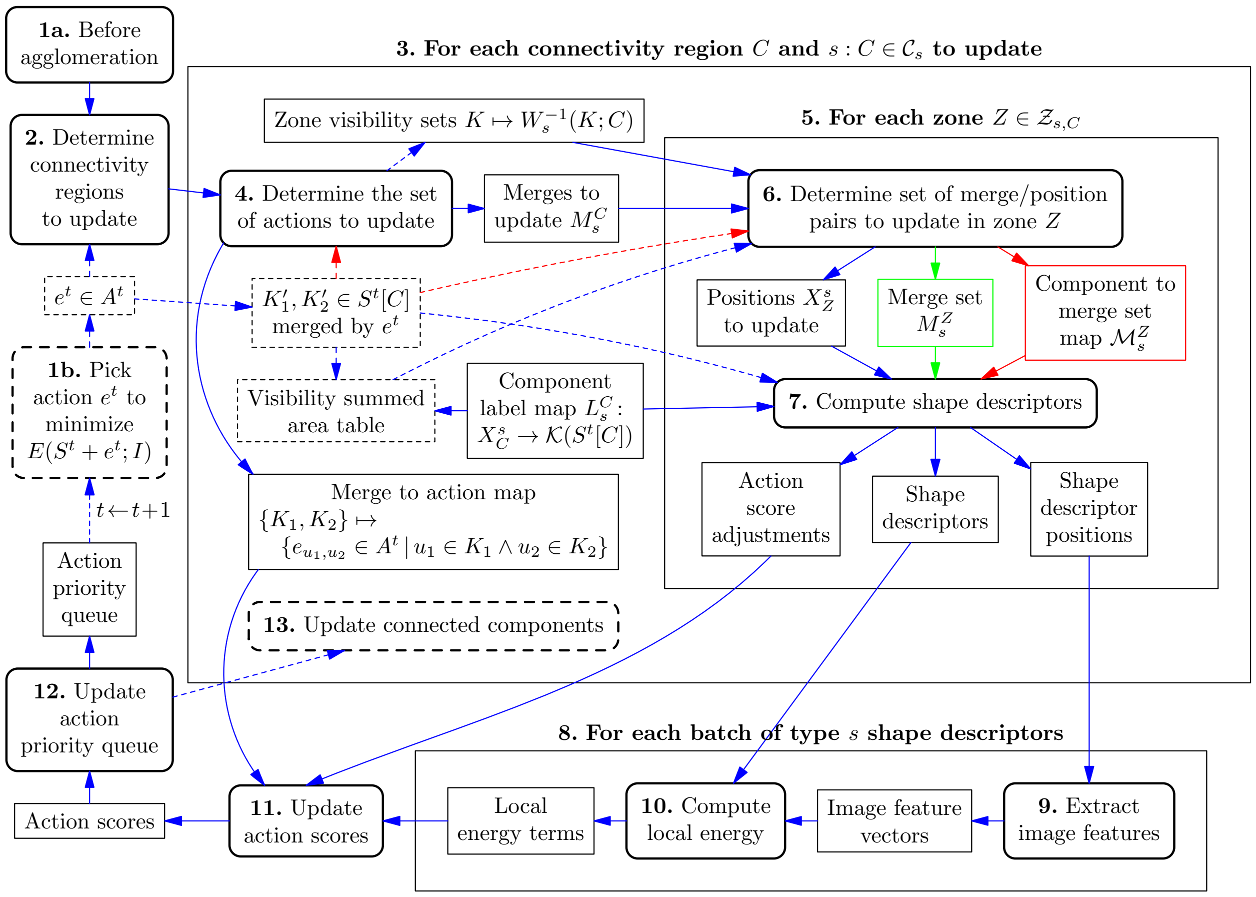
- Because the image feature representation $\phi(x; I)$ is typically expensive to compute, and the same feature is used for computing $E_s(x; S; I)$ for many different candidate segmentations S, we precompute the image features for all positions x and store the feature vectors in a giant 4-D array. In practice the maximum volume size that can be agglomerated is limited by the available memory for storing the precomputed image feature array.

Parallel pipeline for updating action scores

The pipeline for updating action scores is shown in fig. 7.10. The same overall flow of control and data is used both (a) to compute the initial $\Delta^{+e}_{S^0} E(S^0; I)$ scores for all actions $e \in A^0$ prior to agglomeration, and (b) to incrementally update the $\Delta^{+e}_{S^t} E(S^t; I)$ scores from the prior agglomeration step by adding $\Delta^{+e,+e^t}_{S^t} E(S^t; I)$ to reflect the agglomeration action e^t chosen. At a high level, it consists of the following operations:

- **Steps 2–6:** Preprocessing to determine the set of (x, e) position/action pairs for which we must compute shape descriptors $r_s(x; S^t)$, $r_s(x; S^t + e)$, and in the incremental case $r_s(x; S^t + e^t)$ and $r_s(x; S^t + e^t + e)$. This preprocessing is where connectivity region-based pruning (theorem 7.2), graph-based pruning (theorem 7.3), visibility-based pruning (theorem 7.4), and zone-based pruning (theorem 7.6) applies.

- **Step 7:** Computation of shape descriptors $r_s(x; S^t)$, $r_s(x; S^t + e)$, and in the incremental case $r_s(x; S^t + e^t)$ and $r_s(x; S^t + e^t + e)$ for the necessary (x, e) position/action pairs. According to descriptor-based pruning (theorem 7.1), we determine which local energy terms must be computed.



1a. Before agglomeration
2. Determine connectivity regions to update
$e^t \in A^t$
1b. Pick action e^t to minimize $E(S^t + e^t; I)$
$t \leftarrow t+1$
Action priority queue
12. Update action priority queue
Action scores
3. For each connectivity region C and $s : C \in \mathcal{C}_s$ to update
Zone visibility sets $K \mapsto W_s^{-1}(K; C)$
4. Determine the set of actions to update
Merges to update M_s^C
$K'_1, K'_2 \in S^t[C]$ merged by e^t
Visibility summed area table
Component label map $L_s^C : X_C^s \to \mathcal{K}(S^t[C])$
Merge to action map $\{K_1, K_2\} \mapsto \{e_{u_1,u_2} \in A^t \mid u_1 \in K_1 \wedge u_2 \in K_2\}$
13. Update connected components
5. For each zone $Z \in \mathcal{Z}_{s,C}$
6. Determine set of merge/position pairs to update in zone Z
Positions X_Z^s to update
Merge set M_s^Z
Component to merge set map $\mathcal{M}_s^Z$
7. Compute shape descriptors
Action score adjustments
Shape descriptors
Shape descriptor positions
8. For each batch of type s shape descriptors
11. Update action scores
Local energy terms
10. Compute local energy
Image feature vectors
9. Extract image features

Figure 7.10: Pipeline for updating CELIS action $\Delta_{S^t}^{+e} E(S^t; I)$ scores. Arrows show the flow of data (indicated by rectangles) and control (indicated by rounded rectangles). The same pipeline is used both to compute the $\Delta_{S^0}^{+e} E(S^0; I)$ scores *non-incrementally* (starting at 1a) at the start of agglomeration, and to *incrementally* (starting at 1b) update the $\Delta_{S^t}^{+e} E(S^t; I)$ scores from the previous step by adding $\Delta_{S^t}^{+e,+e^t} E(S^t; I)$. Dashed lines indicate steps and dependencies that apply only to the incremental case. Green or red lines indicate steps and dependencies that apply only to pairwise or center-based shape descriptor specifications s, respectively. To limit the complexity of the diagram, the dependencies on the persistent data structures shown in fig. 7.9 are omitted. In the non-incremental case (1a), the set of connectivity regions to update will be the full set $\cup_s \mathcal{C}_s$ and the set of merges to update (determined by step 2) will be the full active set $A^0[C]$. In the incremental case (1b), the zone visibility sets are updated in step 4 per theorem 7.5 to reflect the merge of K'_1 and K'_2, *prior to* computing shape descriptors, to allow the conditions of theorem 7.6 to be checked conveniently; the connected components (represented as disjoint sets of initial supervoxels $\mathcal{K}(S^0)$), which affect the component label map $X_C^s \to \mathcal{K}(S^t[C])$, are updated in step 13 only *after* computing shape descriptors, because the incremental update depends on computing shape descriptors $r_s(x; S^t)$ and $r_s(x; S^{t+1})$ based on both the existing and next segmentation state.

- **Steps 9–10:** Computation of local energy terms needed to compute non-zero $\Delta_{S^t}^{+e} E_s(x; S^t; I)$ terms or, in the incremental case, non-zero $\Delta_{S^t}^{+e,+e^t} E_s(x; S^t; I)$ terms.
- **Steps 11–12:** Updating the global action scores based on the local energy changes.

The pipeline executes using all available processors on a single machine, through the use of a thread pool. The low-level details of the pipeline steps are as follows:

1. **(a) Before agglomeration/(b) Pick action e^t to minimize $E(S^t + e^t; I)$.**
2. **Determine connectivity regions to update.** In the non-incremental case, all connectivity regions $C \in \cup_s \mathcal{C}_s$ must be processed. In the incremental case, per theorem 7.2, only connectivity regions in $C \in \{C \in \cup_s \mathcal{C}_s \mid e^t \in A^t[C]\}$ must be processed. Because we maintain this set of connectivity regions for each action $e \in A^t$, there is only constant (low) overhead for each connectivity region *processed*, and no cost for connectivity regions not processed.
3. **Per-connectivity region processing:** The connectivity regions that must be updated are processed in parallel. While most processing is actually done at the finer per-zone granularity, certain information is computed per-connectivity-region and per associated shape descriptor $s : C \in \mathcal{C}_s$:
   - **Component label map:** a 3-D array that maps positions in the space X_C^s to components in $\mathcal{K}(S^t[C])$, represented by integer identifiers. This is computed by mapping the supervoxel identifier for each position $x \in X_C^s$, which is precisely what is stored to represent S^0, to the corresponding component based on the map from global supervoxels $\mathcal{K}(S^0)$ to connected components $\mathcal{K}(S^t[C])$ in C that we maintain.
   - $K_1', K_2' \in \mathcal{K}(S^t[C])$ **merged by** e^t **(incremental only):** We also use the global supervoxel to local connected component map to translate the action e^t to the pair of components $K_1', K_2' \in \mathcal{K}(S^t[C])$ for which it is a supervoxel merge. Note that it is guaranteed that e^t is a supervoxel merge in C because in the incremental case we only process connectivity regions C for which $e^t \in A^t[C]$.
   - **Visibility summed area table (incremental only):** We compute a single summed area table for $K_1' \cup K_2'$ based on the component label map according to algorithm 7.1.
4. **Determine the set of actions to update.** In this step, for a given connectivity region, we determine the set of actions $e \in A^t[C]$ for which me may *potentially* need to compute shape descriptors $r_s(x; S^t)$, $r_s(x; S^t + e)$, and in the incremental case $r_s(x; S^t + e^t + e)$, according to theorem 7.2. Note that these actions will additionally be filtered in step 6 on a per-zone basis. In the non-incremental case, and also in the incremental case for pairwise s, all actions $e \in A^t[C] - \{e^t\}$ must be (potentially) processed. In the

incremental case for center-based s, only actions $e \neq e^t$ incident to e^t in $S^t[C]$ must be processed, per theorem 7.3. Because we maintain the set of actions incident to each component in $\mathcal{K}(S^t[C])$, computing this set requires only constant time per action to be processed.

Outputs:

- **Merges to update:** the set M_C of *merges*, i.e. pairs of components $\{K_1, K_2\} \subset \mathcal{K}(S^t[C])$ (represented as pairs of integer component identifiers) merged by the actions to be processed. Note that the same pair of components may correspond to more than one action $e \in A^t$, but computation of shape descriptors depends only on the pair of components merged by the action. We therefore use the component representation to avoid redundant computations.
- **Merges to action map:** a mapping from each component pair in M_C to the set of one or more corresponding actions:

  $$\{K_1, K_2\} \in M_C \mapsto \big\{e_{u_1,u_2} \in A^t \,\big|\, u_1 \in K_1 \wedge u_2 \in K_2\big\}.$$

  This is implemented as a hash table mapping pairs of component identifiers to lists of actions. Because energy terms will be locally computed per component pair rather than per action, but globally we maintain per-action scores, this mapping is used to efficiently update all corresponding global per-action scores according to each local per-component-merge score.
- **Zone visibility sets (incremental only):** The zone visibility sets, which are represented as a mapping from integer component identifiers to bit vectors, are updated in this step per theorem 7.5 to reflect the merge of K_1' and K_2' in $(S^t + e^t)[C]$. This simply involves taking the bit-wise OR of the bit vectors.

5. **Per-zone processing**: It is at the granularity of zones that shape descriptor computation actually happens. All zones are processed independently, and in parallel (zones of separate connectivity regions are also processed in parallel) to the extent that there are available cores. Zone processing does, however, depend on certain read-only data structures that are computed per-connectivity region and shared by all zones, including the component label map L_s^C, the set of merges M_s^C to potentially update, and in the incremental case, the visibility summed area table.

6. **Determine set of merge/position pairs to update in zone Z.** The purpose of this step is to finish preprocessing in order to finalize the set of (x, e) position/merge pairs for which we will compute shape descriptor changes. Per theorem 7.6 and definition 7.29, we filter the set of per-connectivity-region merges to update M_s^C based on the zone visibility sets:

$$M_s^Z := \big\{\{K_1, K_2\} \in M_s^C \,\big|\, Z \in W_s^{-1}(K_1^*; C) \cap W_s^{-1}(K_2^*; C)\big\},$$

where in the non-incremental case $K^* := K$ but in the incremental case K^* is the component $K \in \mathcal{K}((S^t + e^t)[C])$ that contains K. Note that in the implementation this happens transparently because the zone visibility bit vectors that are maintained for each component identifier are updated in the incremental case in step 4 to reflect $S^{t+1} = S^t + e^t$. We output either this flat merge set directly or a table of merges incident to each component in $(S^t + e^t)[C]$, depending on whether s is pairwise or center-based.

Outputs:

- **Positions X_Z^s to update:** In the non-incremental case, the set of positions to update is simply $X_Z^s := Z$. In the incremental case, we apply algorithm 7.1 to the visibility summed area table precomputed in step 3 in order to determine the subset of positions $X_Z^s \subseteq Z$ that must be updated. The time complexity is linear in $|Z|$. To limit preprocessing overhead, we only use the first condition of theorem 7.4 and do not test the more complicated second condition.
- **Merge set M_s^Z (pairwise s only):** In the case of a pairwise shape descriptor specification s, we can perform no further merge pruning, and must process all merges in M_s^Z.
- **Component to merge set map $\mathcal{M}_s^Z$ (center-based s only):** In the case of a center-based shape descriptor specification s, the subset of merges in M_s^Z that must be processed for a given position x depends on $K(x; S^t[C])$ in the non-incremental case, or $K(x; (S^t + e^t)[C])$ in the incremental case. We therefore compute a table $\mathcal{M}_s^Z : \mathcal{K}(S^t[C]) \to 2^{M_s^Z}$ that maps
$$K \in \mathcal{K}(S^t[C]) \mapsto \big\{\{K_1, K_2\} \in M_s^Z \,\big|\, K^* \in \{K_1^*, K_2^*\}\big\},$$
where K^* is defined as above. In the non-incremental case, each merge in M_s^Z will occur exactly twice in the table. In the incremental case, each merge will occur exactly 3 times in the table, because every merge in M_s^Z is necessarily incident in $S^t[C]$ to (K_1', K_2').

7. **Compute shape descriptors:** computation of shape descriptors $r_s(x; S^t)$, $r_s(x; S^t + e)$, and in the incremental case $r_s(x; S^t + e^t)$ and $r_s(x; S^t + e^t + e)$ for all (x, e) position/merge pairs determined in step 6. To abstract the difference between pairwise and center-based descriptors, in the case of pairwise s, we define $\mathcal{M}_s^Z : \mathcal{K}(S^t[C]) \to 2^{M_s^Z}$ as the constant function $K \mapsto M_s^Z$. In the non-incremental case, we invoke ComputeDescriptorChanges$(s, X_Z^s, L_s^C, \mathcal{M}_s^Z)$ defined in algorithm 7.3. In the incremental case, we invoke ComputeDescriptorChangesIncremental$(s, \{K_1', K_2'\}, X_Z^s, L_s^C, \mathcal{M}_s^Z$ defined in algorithm 7.4.[4]

Outputs:

[4]We also give the pseudocode for an alternative, more restrictive form, of descriptor-based pruning in algorithm 7.5, that ensures all action score adjustments for a supervoxel merge e correspond to

- **Action score adjustments:** the list of $\langle \{K_1, K_2\}, x, r^-, r^+ \rangle$ tuples specifying updates to the global action scores, implicitly associated with a particular shape descriptor specification s. Each update in the list applies to the one or more global actions that are supervoxel merges of K_1 and K_2 in $S^t[C]$, and corresponds to subtraction of $\hat{E}_s\left(r^-; \phi(x; I)\right)$ and addition of $\hat{E}_s\left(r^+; \phi(x; I)\right)$. Specifically, if we let U denote the aggregate set of all action score adjustments $\langle s, \{K_1, K_2\}, x, r^-, r^+ \rangle$, then we have

$$\Delta_{S^{t+1}}^{+e_{u_1,u_2}} E(S^{t+1}; I) = \Delta_{S^t}^{+e_{u_1,u_2}} E(S^t; I) \tag{7.4}$$

$$+ \sum_{\langle s, \{K_1,K_2\}, x, r^-, r^+ \rangle \in U : u_1 \in K_1 \wedge u_2 \in K_2} \left[\hat{E}_s\left(r^+; \phi(x; I)\right) - \hat{E}_s\left(r^-; \phi(x; I)\right) \right]. \tag{7.5}$$

  We represent the connected components K_1 and K_2 by their corresponding integer identifiers. The same shape descriptors r^- and/or r^+ may occur in multiple action score adjustments, e.g. if they are equal to $r_s(x; S^t)$ or $r_s(x; S^t + e^t)$. To avoid redundant storage in memory and redundant evaluation of the local energy model, we do not directly specify x, r^-, and r^+ in our representation of the action score adjustments list. Instead, we specify r^- and r^+ as integer offsets i^- and i^+ into the list of **shape descriptors** and **shape descriptor positions** also output by this step.

- **Shape descriptors/Shape descriptor positions:** equal length lists specifying the non-redundant shape descriptors/position pairs required by at least one action score adjustment. The lists are constructed in such a way that the $\langle r, x \rangle$ pairs are guaranteed to be unique. The entries are grouped by position x, meaning that if all $\langle r, x \rangle$ pairs for a given position x are contiguous.

8. **Per-batch processing of shape descriptors:** Evaluation of the local energy model on single shape descriptor/image feature pairs may be significantly more expensive than batch evaluation on multiple such pairs. For example, the matrix-vector multiplication required for typical fully-connected neural network activation can be much more efficiently implemented batch-wise as a matrix-matrix multiplication. We therefore collect the shape descriptor/position pairs output from step 7 into batches up to some maximum batch size, e.g. 256. Because different local energy models are used for each shape descriptor specification s, batches are segregated by specification s. We

9. **Extract image features.** We simply copy the image feature vectors $\phi(x; I)$ for each position x in the list of shape descriptor positions for the current batch from the in-memory precomputed image feature array.

   Output:

$(r_s(x; S^t), r_s(x; S^t + e))$ descriptor pairs, rather than possibly $(r_s(x; S^t), r_s(x; S^t + e^t))$. While this restriction is not necessary for the energy model that we describe in this chapter, it allows our same framework to also be used to compute scores defined over actions directly, i.e. pairs of segmentations, rather than individual segmentations.

  - **Image feature vectors:** temporary array holding the copied image feature vectors contiguous in memory.

10. **Compute local energy.** We evaluate in local energy terms $\hat{E}_s\left(r;v\right)$ for the current batch of shape descriptors r and image feature vectors v.

    Output:

    - **Local energy terms:** the list of local energy scores corresponding to the list of shape descriptors in the current batch.

11. **Update action scores.** In this step, we update the global action scores according to eq. (7.4), using the local energy terms computed in step 10 that are referenced by the action score adjustments computed in step 7. To determine the set of (global) actions that correspond to each pair of local connected components specified in the action score adjustments, we we use the merge to action map computed in step 4 for the connectivity region.

12. **Update action priority queue.** After all updates to global action scores are complete, we must correct the ordering of the action priority queue. When performing the initial action score computation prior to agglomeration, we can simply construct the heap in linear time. In the incremental case, we correct the placement of just the action for which the score was updated.

13. **Update connected components (incremental only).** In the incremental case, after computing the update action scores, we update within each affected connectivity region the disjoint sets data structure over supervoxels and the multigraph over connected components to reflect the merge e^t. We do not perform this update until after updating the action scores because in step 7 we need to compute shape descriptors for the segmentation states S^t and $S^t + e$, which would not be possible after merging e^t.

7.7 Experiments

Datasets

We tested our approach on two challenging electron microscopy datasets of neuropil:

FIB25: A *Drosophila melangaster* medulla neuropil dataset [47, 13] at $8 \times 8 \times 8$ nm resolution collected using Focused Ion Beam Scanning Electron Microscopy (FIB-SEM) [59]. The use of Focused Ion Beam milling to access successive layers of tissue allows extremely fine depth resolution, which can be important in reconstructing the finest neurites, as found in organisms like *Drosophila*. 101 79^3, 42 51^3, a 250^3 and a 500^3 voxel volumes were manually annotated with dense neurite segmentations. The 250^3 voxel volume was used for testing; all others were used for training.

Algorithm 7.2 Computation of a single shape descriptor.

```
Require: $s$ is a shape descriptor specification.
Require: $\mathcal{K}$ is a set of components, represented by integers.
Require: $F\colon Z^3 \to \mathcal{K}$ maps shape descriptor offsets to components.
 1: function ComputeDescriptor(s, F)
 2:     Declare $|s|$-bit vector $r$
 3:     if $s$ is pairwise then
 4:         for $\{a, b\} \in s$ do
 5:             $r^{\{a,b\}} \leftarrow \mathbb{1}[F(a) = F(b)]$
 6:         end for
 7:     else
 8:         $K \leftarrow F(\vec{0})$
 9:         for $\{a, \vec{0}\} \in s$ do
10:             $r^{\{a,\vec{0}\}} \leftarrow \mathbb{1}[F(a) = K]$
11:         end for
12:     end if
13:     return $r$
14: end function
```

J0126: A Songbird neuropil dataset at $15 \times 15 \times 25$ nm resolution collected using Serial Block Face Scanning Electron Microscopy (SBEM) [32]. The use of a diamond knife, rather than a Focused Ion Beam, to access succesive layers of tissue allows larger blocks of tissue to be imaged at higher speed. Out of the complete 500 gigavoxel dataset, six $256 \times 256 \times 128$ voxel spatially-separated volumes were manually annotated with dense neurite segmentations. Four of these volumes were used for training, and two for testing agglomeration.

These datasets reflect state-of-the-art techniques in sample preparation and imaging for large-scale neuron reconstruction. We assume that the data distribution is invariant with respect to axis-aligned rotation and reflection in the x-y imaging plane and with respect to reflection about the section axis z. We therefore obtained a factor of 16 increase in the amount of training data by augmenting the original data with all possible transformed versions.

Boundary classification and oversegmentation

To obtain image features and an oversegmentation to use as input for agglomeration, we trained a convolutional neural network to predict, based on a $35 \times 35 \times 9$ voxel image context region, whether the center voxel is part of the same neurite as the adjacent voxel in each of the x, y, and z directions, as in prior work. [81] The network architecture is shown in table 7.1. We trained the network using stochastic gradient descent with log loss. Using these connection affinities, we applied a watershed algorithm [88, 89] to obtain an (approximate) oversegmentation. On J0126, we used parameters $T_l = 0.9999$, $T_h = 0.99$, and $T_e = 0.03$; on

Algorithm 7.3 Computation of shape descriptor changes (non-incremental case). The result is (a) a stream of position/shape descriptors pairs produced by calls to **EmitDescriptor**(x, r), which returns the stream position; (b) a separate stream of score adjustments produced by calls to **EmitScoreAdjustment**(m, i^-, i^+) that associate a merge $m = \{K_1, K_2\}$ with a negative and positive energy contribution corresponding to previously emitted shape descriptors at stream positions i^- and i^+, respectively. The computation of individual shape descriptors is shown in algorithm 7.2.

```
Require: $X \subset \mathbb{Z}^3$ is a set of positions.
Require: $L\colon X \to \mathcal{K}$ maps positions in $X$ to components in $\mathcal{K}$.
Require: $\mathcal{M}\colon \mathcal{K} \to 2^{[\mathcal{K}]^2}$ maps components in $\mathcal{K}$ to sets of merges.
 1: function ComputeDescriptorChanges($s$, $X$, $L$, $\mathcal{M}$)
 2:     Declare array $\psi\colon \mathcal{K} \to \mathcal{K}$
 3:     for $K \in \mathcal{K}$ do
 4:         $\psi(K) \leftarrow K$                                  ▷ Initialize $\psi$ to the identity map.
 5:     end for
 6:     for $x \in X$ do
 7:         $r \leftarrow$ ComputeDescriptor($s, c \mapsto L(x+c)$)
 8:         $i \leftarrow -1$                                     ▷ $-1$ represents an invalid index
 9:         for $\{K_1, K_2\} \in \mathcal{M}(L(x))$ do
10:             $\psi(K_2) \leftarrow K_1$
11:             $r_e \leftarrow$ ComputeDescriptor($s, c \mapsto \psi(L(x+c))$)
12:             $\psi(K_2) \leftarrow K_2$                          ▷ Restore $\psi$ to identity map.
13:             if $r \neq r_e$ then
14:                 if $i = -1$ then $i \leftarrow$ EmitDescriptor($x, r$)
15:                 $i_e \leftarrow$ EmitDescriptor($x, r_e$)
16:                 EmitScoreAdjustment($\{K_1, K_2\}, i, i_e$)
17:             end if
18:         end for
19:     end for
20: end function
```

Table 7.1: Network architecture used for oversegmentation and image features.

Layer	Input	Transform	Output	# parameters	Dropout (p)
1	$1 \times 35 \times 35 \times 9$	$5 \times 5 \times 1$ convolution, ReLU	$64 \times 31 \times 31 \times 9$	$64 \cdot (5^2 + 1)$	0.9
2	$64 \times 31 \times 31 \times 9$	$5 \times 5 \times 5$ convolution, ReLU	$64 \times 27 \times 27 \times 5$	$64 \cdot (64 \cdot 5^3 + 1)$	0.9
3	$64 \times 27 \times 27 \times 5$	$2 \times 2 \times 1$ max pooling	$64 \times 14 \times 14 \times 5$		0.9
4	$64 \times 14 \times 14 \times 5$	$5 \times 5 \times 5$ convolution, ReLU	$64 \times 10 \times 10 \times 1$	$64 \cdot (64 \cdot 5^3 + 1)$	0.9
5	$64 \times 10 \times 10 \times 1$	$2 \times 2 \times 1$ max pooling	$64 \times 5 \times 5 \times 1$		0.9
6	$64 \times 5 \times 5 \times 1$	Fully-connected ReLU	512	$512 \cdot (64 \cdot 5^2 + 1)$	0.5
7	512	Fully-connected logistic	3	$3 \cdot (512 + 1)$	

Algorithm 7.4 Computation of shape descriptor changes (incremental case).

```
Require: $\{K'_1, K'_2\} \subseteq \mathcal{K}$ is a merge.
 1: function ComputeDescriptorChangesIncremental($s$, $\{K'_1, K'_2\}$, $X$, $L$, $\mathcal{M}$)
 2:     Declare arrays $\psi, \psi' \colon \mathcal{K} \to \mathcal{K}$
 3:     for $K \in \mathcal{K}$ do
 4:         $\psi(K), \psi'(K) \leftarrow K$                ▷ Initialize $\psi$ and $\psi'$ to the identity map.
 5:     end for
 6:     $\psi'(K'_2) \leftarrow K'_1$
 7:     for $x \in X$ do
 8:         $r \leftarrow$ ComputeDescriptor($s, c \mapsto L(x+c)$)
 9:         $r_{e'} \leftarrow$ ComputeDescriptor($s, c \mapsto \phi'(L(x+c))$)
10:         $i, i_{e'} \leftarrow -1$                ▷ $-1$ represents an invalid index
11:         for $\{K_1, K_2\} \in \mathcal{M}(L(x))$ do
12:             $\psi(K_2) \leftarrow K_1$
13:             $J'_1 \leftarrow \psi'(K'_1)$, $J'_2 \leftarrow \psi'(K'_2)$
14:             if $K_2 \in \{K'_1, K'_2\}$ then
15:                 $\psi'(K_1) \leftarrow \psi'(K_2)$
16:             else
17:                 $\psi'(K_2) \leftarrow \psi'(K_1)$
18:             end if
19:             $r_e \leftarrow$ ComputeDescriptor($s, c \mapsto \psi(L(x+c))$)
20:             $r_{e,e'} \leftarrow$ ComputeDescriptor($s, c \mapsto \psi'(L(x+c))$)
21:             $\psi(K_2) \leftarrow K_2$                ▷ Restore $\psi$ to identity map.
22:             $\psi'(K_1) \leftarrow J'_1$, $\psi'(K_2) \leftarrow J'_2$                ▷ Restore $\psi'$ to initial value.
23:             if $r_e \neq r_{e,e'}$ then
24:                 if $r_{e'} \neq r_{e,e'}$ then
25:                     if $i_{e'} = -1$ then $i_{e'} \leftarrow$ EmitDescriptor($x, r_{e'}$)
26:                     $i_{e,e'} \leftarrow$ EmitDescriptor($x, r_{e,e'}$)
27:                     EmitScoreAdjustment($\{K_1, K_2\}, i_{e'}, i_{e,e'}$)
28:                 end if
29:                 if $r \neq r_e$ then
30:                     if $i = -1$ then $i \leftarrow$ EmitDescriptor($x, r$)
31:                     $i_e \leftarrow$ EmitDescriptor($x, r_e$)
32:                     EmitScoreAdjustment($\{K_1, K_2\}, i_e, i$)  ▷ Note the order of $i_e$ and $i$.
33:                 end if
34:             else if $r \neq r_{e'}$ then
35:                 if $i = -1$ then $i \leftarrow$ EmitDescriptor($x, r$)
36:                 if $i_{e'} = -1$ then $i_{e'} \leftarrow$ EmitDescriptor($x, r_{e'}$)
37:                 EmitScoreAdjustment($\{K_1, K_2\}, i_{e'}, i$)      ▷ Note the order of $i_{e'}$ and $i$.
38:             end if
39:         end for
40:     end for
41: end function
```

Algorithm 7.5 Alternative computation of shape descriptor changes (incremental case). In order to support a alternative *discriminative* model $\tilde{E}_s\left(r_s(x;S), r_s(x;S+e); \phi(x;I)\right)$ that explicitly scores shape descriptor *changes* due to candidate actions e rather than individual shape descriptors, we must ensure that all score adjustments correspond to such a pair. This requires that in algorithm 7.3 we use the following more restrictive form of pruning in place of lines 23 to 38.

```
1: if $r \neq r_{e'} \vee r_e \neq r_{e,e'}$ then
2:     if $r_{e'} \neq r_{e,e'}$ then
3:         if $i_{e'} = -1$ then $i_{e'} \leftarrow$ EmitDescriptor$(x, r_{e'})$
4:         $i_{e,e'} \leftarrow$ EmitDescriptor$(x, r_{e,e'})$
5:         EmitScoreAdjustment$(\{K_1, K_2\}, i_{e'}, i_{e,e'})$
6:     end if
7:     if $r \neq r_e$ then
8:         if $i = -1$ then $i \leftarrow$ EmitDescriptor$(x, r)$
9:         $i_e \leftarrow$ EmitDescriptor$(x, r_e)$
10:        EmitScoreAdjustment$(\{K_1, K_2\}, i_e, i)$        ▷ Note the order of $i_e$ and $i$.
11:    end if
12: end if
```

FIB25, we used parameters $T_l = 0.94$, $T_h = 0.94$, and $T_e = 0.01$. On both datasets we used the minimum segment size threshold $T_s = 1000$ voxels.

Energy model architecture

We used five types of 512-dimensional shape descriptors: three pairwise descriptor types with 9^3, 17^3, and 33^3 bounding boxes, and two center-based descriptor types with 17^3 and 33^3 bounding boxes, respectively. The connectivity positions within the bounding boxes for each descriptor type were sampled uniformly at random.

We used the 512-dimensional fully-connected penultimate layer output of the low-level classification convolutional neural network as the image feature vector $\phi(x;I)$. For each shape descriptor type s, we used the following architecture for the local energy model $E_s(r;f)$, as shown in fig. 7.11: we concatenated the shape descriptor vector and the image feature vector to obtain a 1024-dimensional input vector. We used two 512- or 1024-dimensional fully-connected rectified linear hidden layers, followed by a logistic output unit, and applied dropout (with $p = 0.5$) after the last hidden layer. While this effectively computes a score from a raw image patch and a shape descriptor, by segregating expensive convolutional image processing that does not depend on the shape descriptor, this architecture allows us to benefit from pre-training and precomputation of the intermediate image feature vector $\phi(x;I)$ for each position x. Training for both the energy models and the boundary classifier was performed using asynchronous SGD using a distributed architecture. [30]

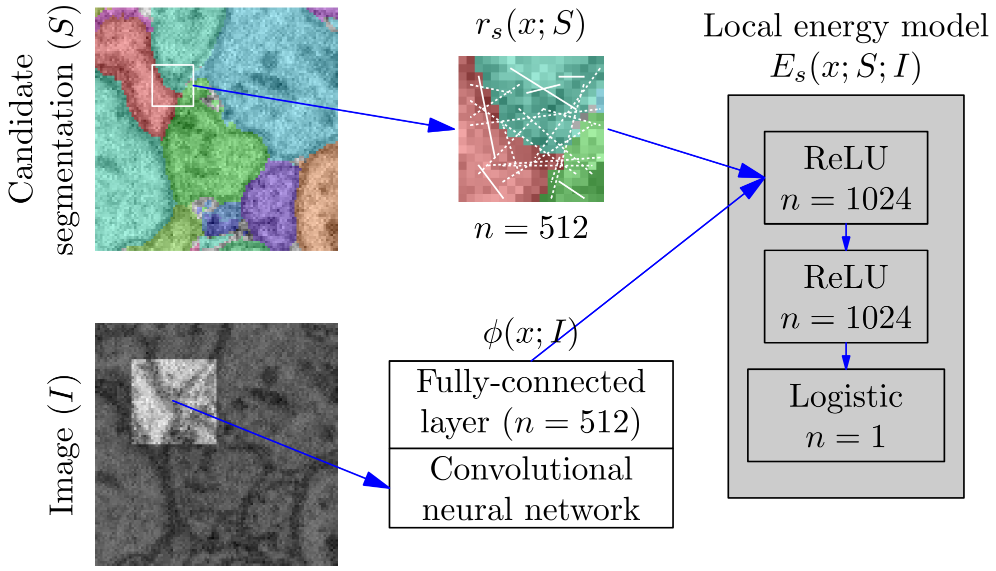

Candidate segmentation (S)
$r_s(x;S)$
$n = 512$
Local energy model $E_s(x;S;I)$
ReLU $n = 1024$
ReLU $n = 1024$
Logistic $n = 1$
Image (I)
$\phi(x;I)$
Fully-connected layer $(n = 512)$
Convolutional neural network

Figure 7.11: Architecture of the specific local energy models used for experiments. For each shape descriptor type/scale s, an identical architecture is used to define the corresponding local energy model E_s, but the learned parameters are distinct.

Evaluation

We compared our method to the state-of-the-art agglomeration method GALA [68], which is designed to use as image features voxel-wise predicted probabilities of boundary vs. cell interior. [5] To obtain such probabilities from our low-level convolutional neural network classifier, which predicts edge affinities *between* adjacent voxels rather than per-voxel predictions, we compute for each voxel the minimum connection probability to any voxel in its 6-connectivity neighborhood, and treat this as the probability/score of it being cell interior.

For each of the two datasets, we trained and tested both GALA and CELIS, using the same initial oversegmentations. Consistent with prior work [82, 23, 68], we evaluated the agglomeration results on each test volume by computing two measures of segmentation consistency relative to the ground truth: Variation of Information [65] and Rand F_1 score, defined as the F_1 classification score over connectivity between all voxel pairs within the volume. The former has the advantage of weighing segments linearly in their size rather than quadratically.

Because any agglomeration method is ultimately limited by the quality of the initial oversegmentation, we also computed the accuracy of an *oracle* agglomeration policy that greedily optimizes the error metric directly. (Computing the true globally-optimal agglomeration under either metric is intractable.) This serves as an (approximate) upper bound

[5] GALA also supports multi-channel image features, potentially representing predicted probabilities of additional classes, such as mitochondria, but we did not make use of this functionality as we did not have training data for additional classes for both datasets.

Table 7.2: Accuracy of segmentations obtained by agglomeration, measured by Variation of Information (lower is better) and Rand F_1 score (higher is better). Both GALA and our method CELIS depend on a single threshold parameter that trades off false merges against false splits; to simplify the presentation we report for each metric just the best possible score over all thresholds. Because both methods rely on randomness during training (random forest learning in the case of GALA, sampling of connectivity pairs for shape descriptors and random initialization of energy model weights in the case of CELIS), we report the mean and standard deviation over multiple independently-trained agglomeration policies. For CELIS, we show results based on either just the pairwise 9^3 shape descriptor type, or all five shape descriptor types. 512-d or 1024-d indicates the dimensionality of the two hidden layers in each energy model.

	J0126 (Volume 1)		J0126 (Volume 2)		FIB25	
	VI	Rand F_1	VI	Rand F_1	VI	Rand F_1
Initial oversegmentation	1.063972	0.918597	1.434858	0.803590	4.046593	0.233730
GALA	0.339979	0.974334	0.439647	0.948633	1.295051	**0.915477**
$n = 10$	± 0.003048	± 0.000034	± 0.003618	± 0.000336	± 0.012717	± 0.003607
CELIS (512-d, 9^3 only)	0.330849	0.975285	0.415100	0.964628	1.301229	0.910730
$n = 3$	± 0.000326	± 0.000001	± 0.001779	± 0.000000	± 0.001541	± 0.000001
CELIS (1024-d, 9^3 only)	0.330741	0.975286	0.414740	0.964628	1.299110	0.910901
$n = 3$	± 0.000175	± 0.000001	± 0.001587	± 0.000000	± 0.000306	± 0.000243
CELIS (512-d, all descriptors)	0.330263	0.975287	**0.392324**	0.966862	1.279776	0.911253
$n = 3$	± 0.000029	± 0.000000	± 0.000001	± 0.000000	± 0.001954	± 0.000007
CELIS (1024-d, all descriptors)	**0.330220**	**0.975287**	0.392335	**0.966862**	**1.277726**	0.911223
$n = 3$	± 0.000032	± 0.000001	± 0.000018	± 0.000000	± 0.001029	± 0.000043
Oracle	0.278919	0.979529	0.338502	0.973650	0.959395	0.954735

that is useful separating the error due to agglomeration from the error due to the initial oversegmentation.

7.8 Results

Table 7.2 shows the segmentation accuracy results. As measured by Variation of Information, the metric that both GALA and CELIS are trained to optimize, CELIS consistently outperformed GALA on all 3 test volumes, reducing the remaining error of GALA relative to the oracle agglomeration by 23% on average.[6] The Rand F_1 scores show a similar pattern, except on the FIB25 volume, where GALA outperforms CELIS. The addition of multiple shape descriptor scales provided a moderate but consistent improvement in performance. The size of the hidden layers used for the local energy models did not significantly affect

[6]We also evaluated several other recent methods, including deep CNN boundary classification alone [23], segmentation fusion [83] (which we found to outperform the similar method SOPNET [37]), and globally-optimal multicut [4], but none were competitive with GALA or CELIS.

performance. Global training turned out not to provide any significant improvement over local training.

We also measured the effectiveness of each of the computational pruning tricks described in section 7.5. Essentially the entire computational cost of CELIS is in computing shape descriptors and evaluating the local energy models; the cost of performing the pruning and other preprocessing turns out to be negligible (less than 1%). Therefore the savings in descriptors processed correspond directly to savings in overall runtime. With pruning, computation of shape descriptors accounted for about 20% of the cost; the remainder was spent evaluating the energy model. Without it, the cost would be several orders of magnitude higher. The results are shown in fig. 7.12.

7.9 Discussion

CELIS is a flexible framework for learning to segment images based on optimization of a learned global energy model. It achieves state-of-the-art performance on two challenging connectomics datasets. Using the novel binary shape descriptor representation, it can capture rich prior information about shape configurations. The cost of this rich representation is that we must forgo exact global optimization, but we do define an efficient local search procedure. Unlike prior agglomeration methods [83, 4, 37, 49, 13], including GALA [68], that are limited to reasoning about region pairs, our energy model reflects the full combinatorial nature of optimizing dense segmentation. End-to-end training of the complete pipeline, including the convolutional neural network for computing image features, is a promising future direction.

The key property of the binary shape descriptor is that it is very stable with respect to small perturbations of segment boundaries, compared to a simple boundary mask. Compared to existing log-polar histogram-based descriptors like 2-D shape context [8] and 3-D variants thereof [13], a center-based binary shape descriptor represents essentially the same information with some added noise due to sampling. The advantage is a significant reduction in computational cost, particularly at large scales, since we need only access the (relatively few) sampled positions. We know of no obvious way to define a histogram-based analogue of a pairwise binary shape descriptor.

While conceptually very similar, segmentation of natural scenes is fundamentally quite different from neurite segmentation: it is complicated by 2-D projection, occlusion and lighting effects, but is made easier by strong local appearance cues for distinguishing segments, and more rigid geometry.

Our work complements prior work on translation-invariant energy models, such as Field of Experts [72] (not conditional, and limited to linear+activation function local potentials) and Regression Tree Fields [51], which are limited to unary and pairwise interactions. Multiscale Fields of Patterns [36] is designed to model a binary pixel-wise labeling rather than a general segmentation, and uses a direct table representation of local potentials, rather than the neural network-based function approximation used in our work in order to support much higher-dimensional patterns.

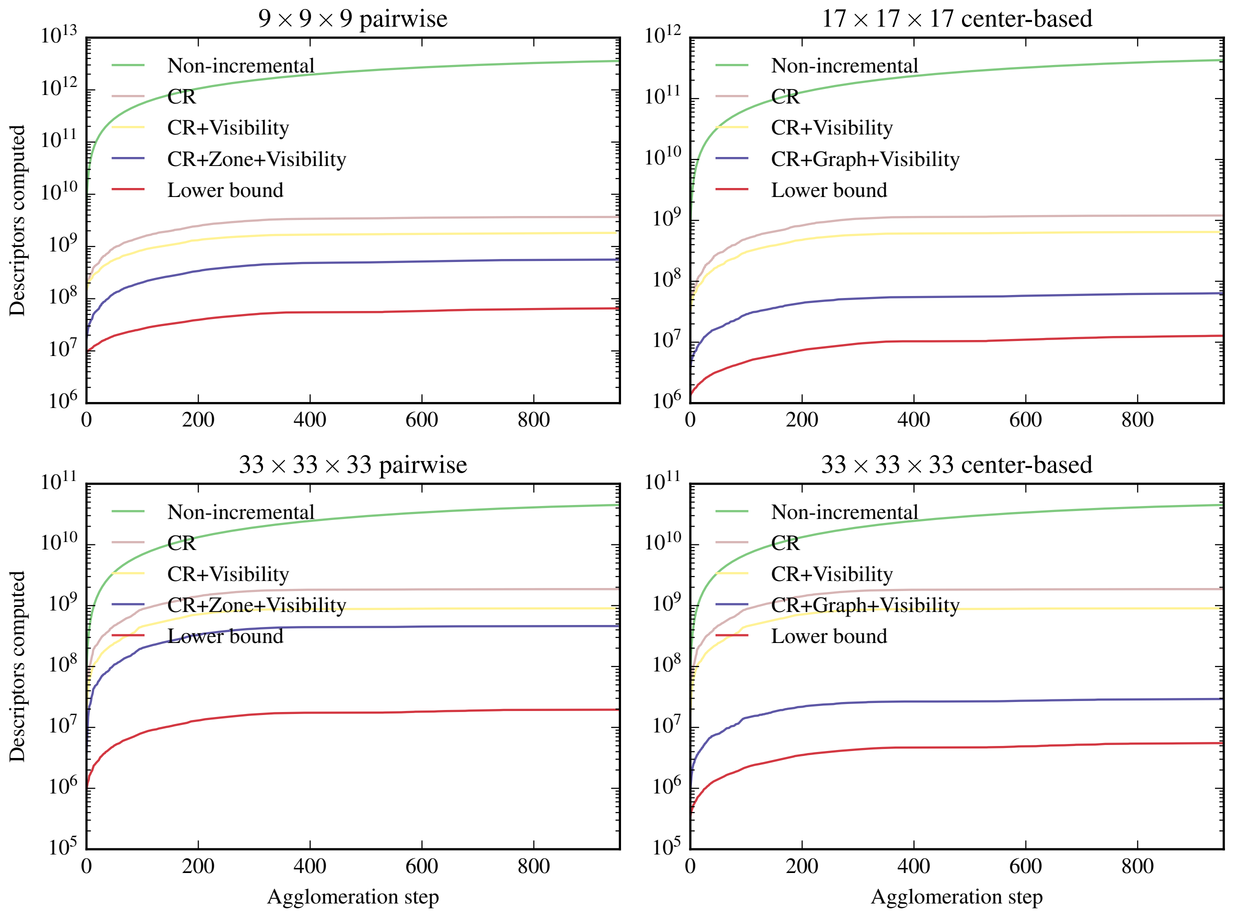


Figure 7.12: Effect of pruning on number of shape descriptors computed. The vertical axis specifies the *cumulative* number of shape descriptors computed during the course of agglomeration, using different combinations of pruning rules. The horizontal axis specifies the agglomeration step t, with $t = 0$ indicating the computation required to *initialize* the energy first derivative terms. *Non-incremental* corresponds a naïve implementation that does no pruning or incremental computation whatsoever. The different combinations of *CR*, *Visibility*, *Zone*, and *Graph* correspond to correspond to applying combinations of theorem 7.2, theorem 7.4, theorem 7.6, and theorem 7.3, respectively. The actual number of descriptors that changed is shown as the lower bound, since in the best case pruning would eliminate the computation of all but these descriptors. This is also the number of evaluations of the energy model performed. If the combination of pruning techniques were perfect, it would exactly match this lower bound. Results are shown for a 100^3 portion of the J0126 dataset.

Our framework also falls within the "learning to search" paradigm, and is quite similar to a single iteration of DAgger [71]. However, a notable property of our agglomeration search space is that the order of actions (merges) does not affect the final result; consequently, we expect that the benefit from more sophisticated reinforcement learning algorithms may be small.

Chapter 8

Conclusion

Densely mapping neuroanatomy, in order to reconstruct circuits of tens or hundreds of thousands of neurons, is a challenging problem at the intersection of neuroscience and computer data processing. The reconstruction of the complete nervous system of C. elegans, [86] a seminal work (mostly) completed in the 1980s, demonstrated that the problem was solvable in principle by collecting electron microscopy image stacks using the Serial Section Transmission Electron Microscopy (ssTEM) , and impractical for the methods used to scale to larger circuits, due to the enormous difficulties of data collection and manual tracing of neuronal processes through the image volume.

Recent advances in volume electron microscopy have greatly reduced the difficulty of data collection. The recent methods of Serial Block Face Scanning Electron Microscopy (SBEM) [32], Focused Ion Beam Scanning Electron Microscopy (FIB-SEM) [59], and Automated Tape-collection Ultramicrotome Scanning Electron Microscopy (ATUM-SEM) [75], in combination with advances such as multi-beam scanning electron microscopy, greatly increase image acquisition rate compared to manual ssTEM imaging, and also greatly improve image quality, specifically by reducing damage to and loss of section, improving alignment, and improving depth resolution. These advances have essentially solved the data collection problem to a sufficient degree that it is no longer a key bottleneck.

The ability to collect large, high quality data volumes has spurred work on automated image analysis methods for the problem, particularly segmentation. Prior work has focused on both low-level boundary classification at the individual voxel level as well as higher-level agglomeration methods that combine together smaller regions obtained from boundary classification. Machine learning has proven crucial for both of these tasks, and significant progress has been made. There has also been extensive prior work on determining appropriate metrics for evaluating the accuracy of automated segmentations. Despite the progress made by prior work, however, image analysis remains a key challenge, which this thesis aims to address.

The sparse, wide architecture we introduced in chapter 5 provides an alternative to conventional dense convolutional neural networks for boundary classification. By relying on simple, parallelizable algorithms that are highly amenable to parallel training, we were able to train models with tens of thousands of convolutional feature maps per layer, several orders of magnitude higher than existing dense convolutional neural networks. Using an optimized GPU implementation based on filter size-specific automatic code generation, we were able to execute the model using essentially the full computational and memory resources of the hardware. By encoding rotational covariance properties directly into the model, we reduced the memory and computational requirements for training by a factor of 48, which dramatically improved scalability. These factors contributed to having a training time measured in hours rather than weeks typical of neural network models; reducing the training time for machine learning is crucial for efficient exploration of the space of hyper-parameters and model architectures.

The EMISAC algorithm introduced in chapter 6 reduces the local discontinuities in image intensity between sections using a coarse-to-fine convex optimization procedure. These discontinuities are a significant issue in the image data obtained using the high-throughput methods of ssTEM and ATUM-SEM: uncorrected, they essentially force the use of 2.5D analysis, in which results are initially computed on a per-section basis and then later joined together. For neuronal processes not running parallel to the sectioning axis, this leads to poor results. We found EMISAC greatly improves statistical measures of isotropy without any apparent loss of detail, and made possible the use of fully 3-D features and analysis, which improved segmentation accuracy.

CELIS, introduced in chapter 7, achieved state-of-the-art accuracy for neurite segmentation. It is a powerful framework for image segmentation based on optimization of a learned deep neural network-based global energy model over segmentations. Using the novel binary shape descriptor representation, this approach captures rich prior information about shape configurations, including interactions between multiple objects. It differs critically from prior agglomeration methods in that it defines an energy over segmentations in a highly general way, rather than modeling only pairwise merge decisions. This generality comes at the cost of greater implementation complexity, but we also prove a collection of properties that allow for incremental evaluation of energy terms, with an overall several order of magnitude reduction in computational cost.

Future Directions

The accuracy of boundary prediction appears to be limited currently by the capacity of the convolutional models that can reasonably be trained. Incorporating the efficient sparse convolutional operations introduced in chapter 5 into deep neural network architectures may provide a way to train sparse models with many more convolutional filters (and parameters) than can efficiently be supported in dense networks.

In our experiments, we reused the final hidden layer output of the convolutional network trained for boundary classification as the image feature representation for CELIS. While this simplified training, end-to-end training of the convolutional network as part of training the local energy models may likely improve performance; the convolutional portion of the local energy model can still remain independent of the shape descriptor in order to allow the same computational savings at test time.

Existing iterative agglomeration methods, including CELIS, are based on a *merge-only* framework of starting from an initial oversegmentation and greedily performing merges. This simplifies the implementation, but is more prone to getting stuck in local optima than a randomized simulated annealing-type procedure that that can also perform splits (equivalent to *undoing* a merge). Prior methods based on an arbitrary learned score over pairwise merge decisions would have no guarantee of convergence if splits are permitted. Because the CELIS agglomeration procedure optimizes a true global objective over segmentations, however, it readily supports simulated annealing. Efficiently tracking the necessary connectivity information within each connectivity region is more complicated when splits are permitted, but it can be done using dynamic graph connectivity data structures. [45, 46, 80].

Significant challenges remain, however. In addition to greater accuracy, reliable measures of confidence are crucial for effectively combining machine and human efforts, and this is something that has so far proven difficult to obtain. Successful circuit reconstruction is also just a first step towards understanding neural computation: analysis of the large-scale circuits that are reconstructed may prove to be an even greater challenge than the reconstruction itself.

Bibliography

[1] Andrew L Alexander, Jay S Tsuruda, and Dennis L Parker. "Elimination of eddy current artifacts in diffusion-weighted echo-planar images: the use of bipolar gradients". In: *Magnetic Resonance in Medicine* 38.6 (1997), pp. 1016–1021.

[2] Jesper LR Andersson, Stefan Skare, and John Ashburner. "How to correct susceptibility distortions in spin-echo echo-planar images: application to diffusion tensor imaging". In: *Neuroimage* 20.2 (2003), pp. 870–888.

[3] Bjoern Andres et al. "3D segmentation of SBFSEM images of neuropil by a graphical model over supervoxel boundaries". In: *Medical image analysis* 16.4 (2012), pp. 796–805.

[4] Bjoern Andres et al. "Globally optimal closed-surface segmentation for connectomics". In: *Computer Vision–ECCV 2012*. Springer, 2012, pp. 778–791.

[5] B. Andres et al. "Segmentation of SBFSEM volume data of neural tissue by hierarchical classification". In: *Pattern recognition* (2008), pp. 142–152.

[6] Samaneh Azadi, Jeremy Maitin-Shepard, and Pieter Abbeel. "Optimization-Based Artifact Correction for Electron Microscopy Image Stacks". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2014.

[7] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. "Correlation Clustering". English. In: *Machine Learning* 56.1-3 (2004), pp. 89–113. ISSN: 0885-6125.

[8] Serge Belongie, Jitendra Malik, and Jan Puzicha. "Shape matching and object recognition using shape contexts". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 24.4 (2002), pp. 509–522.

[9] Daniel R. Berger et al. *SNEMI3D challenge*. `http://brainiac2.mit.edu/SNEMI3D/home`. (Training volume).

[10] S. Beucher and F. Meyer. "The morphological approach to segmentation: the watershed transformation". In: *Mathematical morphology in image processing*. Ed. by E. R. Dougherty. Vol. 34. 1993, pp. 433–481.

[11] Serge Beucher and Christian Lantuéjoul. "Use of watersheds in contour detection". In: *International workshop on image processing, real-time edge and motion detection.* 1979.

[12] Andrew Blake, Rupert Curwen, and Andrew Zisserman. “A Framework for Spatio-Temporal Control in the Tracking of Visual Contours”. In: *Real-time computer vision* 4 (1994), p. 3.

[13] John A Bogovic, Gary B Huang, and Viren Jain. “Learned versus hand-designed feature representations for 3d agglomeration”. In: *arXiv preprint arXiv:1312.6159* (2013).

[14] Yuri Boykov and Vladimir Kolmogorov. “An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 26.9 (2004), pp. 1124–1137.

[15] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.

[16] Kevin L Briggman and Davi D Bock. “Volume electron microscopy for neuronal circuit reconstruction”. In: *Current opinion in neurobiology* 22.1 (2012), pp. 154–161.

[17] Kevin L Briggman and Winfried Denk. “Towards neural circuit reconstruction with volume electron microscopy techniques”. In: *Current Opinion in Neurobiology* 16.5 (2006). Neuronal and glial cell biology / New technologies, pp. 562–570. ISSN: 0959-4388. DOI: `DOI:10.1016/j.conb.2006.08.010`. URL: `http://www.sciencedirect.com/science/article/B6VS3-4KV8T8F-1/2/0d423f67d75acb1f58510df3dfb1a61e`.

[18] Kevin L Briggman, Moritz Helmstaedter, and Winfried Denk. “Wiring specificity in the direction-selectivity circuit of the retina”. In: *Nature* 471.7337 (2011), pp. 183–188.

[19] Thomas Brox and Joachim Weickert. “Nonlinear matrix diffusion for optic flow estimation”. In: *Pattern Recognition*. Springer, 2002, pp. 446–453.

[20] Albert Cardona et al. “An integrated micro-and macroarchitectural analysis of the Drosophila brain by computer-assisted serial section electron microscopy”. In: *PLoS biology* 8.10 (2010), e1000502.

[21] Albert Cardona et al. *Segmented serial section Transmission Electron Microscopy (ssTEM) data set of the Drosophila first instar larva ventral nerve cord (VNC)*. `http://www.ini.uzh.ch/~acardona/data.html`. 2010.

[22] R. Chylinski. *Time-Lapse Photography: A Complete Guide to Shooting, Processing and Rendering Time-Lapse Movies*. Cedar Wings Creative, 2012. ISBN: 9780985375720. URL: `http://books.google.com/books?id=7fDaLPhJB5IC`.

[23] Dan Claudiu Ciresan et al. “Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images”. In: *NIPS*. 2012, pp. 2852–2860.

[24] Adam Coates, Andrej Karpathy, and Andrew Ng. “Emergence of Object-Selective Features in Unsupervised Feature Learning”. In: *Advances in Neural Information Processing Systems 25*. 2012, pp. 2690–2698.

[25] Adam Coates and Andrew Y Ng. “Learning Feature Representations with k-means”. In: *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 561–580.

[26] Adam Coates and Andrew Y Ng. “The importance of encoding versus training with sparse coding and vector quantization”. In: *International conference on machine learning*. Vol. 8. 2011, p. 10.

[27] Thomas H. Cormen et al. *Introduction to algorithms*. Second. The MIT Press, 2001. ISBN: 0262032937.

[28] R Cameron Craddock et al. “Imaging human connectomes at the macroscale”. In: *Nature methods* 10.6 (2013), pp. 524–539.

[29] Franklin C Crow. “Summed-area tables for texture mapping”. In: *ACM SIGGRAPH computer graphics* 18.3 (1984), pp. 207–212.

[30] Jeffrey Dean et al. “Large Scale Distributed Deep Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1223–1231. URL: `http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf`.

[31] Winfried Denk, Kevin L Briggman, and Moritz Helmstaedter. “Structural neurobiology: missing link to a mechanistic understanding of neural computation”. In: *Nature Reviews Neuroscience* 13.5 (2012), pp. 351–358.

[32] Winfried Denk and Heinz Horstmann. “Serial Block-Face Scanning Electron Microscopy to Reconstruct Three-Dimensional Tissue Nanostructure”. In: *PLoS Biol* 2.11 (Oct. 2004), e329. DOI: `10.1371/journal.pbio.0020329`.

[33] Winfried Denk and Heinz Horstmann. “Serial block-face scanning electron microscopy to reconstruct three-dimensional tissue nanostructure”. In: *PLoS biology* 2.11 (2004), e329.

[34] Nick Duffield, Carsten Lund, and Mikkel Thorup. “Priority sampling for estimation of arbitrary subset sums”. In: *Journal of the ACM (JACM)* 54.6 (2007), p. 32.

[35] David S. Dummit and Richard M. Foote. *Abstract Algebra*. Third. John Wiley and Sons, Inc., 2004. ISBN: 0471433349. URL: `https://books.google.com/books?id=znzJygAACAAJ`.

[36] Pedro Felzenszwalb and John G Oberlin. “Multiscale fields of patterns”. In: *Advances in Neural Information Processing Systems*. 2014, pp. 82–90.

[37] Jan Funke et al. “Efficient automatic 3D-reconstruction of branching neurons from EM data”. In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE. 2012, pp. 1004–1011.

[38] John C Haselgrove and James R Moore. “Correction for distortion of echo-planar images used to calculate the apparent diffusion coefficient”. In: *Magnetic Resonance in Medicine* 36.6 (1996), pp. 960–964.

[39] Kenneth J Hayworth et al. “Ultrastructurally smooth thick partitioning and volume stitching for large-scale connectomics”. In: *Nature methods* 12.4 (2015), pp. 319–322.

[40] KJ Hayworth et al. “Automating the Collection of Ultrathin Serial Sections for Large Volume TEM Reconstructions”. In: *Microscopy and Microanalysis* 12.Supplement,S02 (2006), pp. 86–87. DOI: 10.1017/S1431927606066268. eprint: http://journals.cambridge.org/article_S1431927606066268. URL: http://journals.cambridge.org/action/displayAbstract?fromPage=online&aid=456486&fulltextType=PI&fileId=S1431927606066268.

[41] Moritz Helmstaedter. “Cellular-resolution connectomics: challenges of dense neural circuit reconstruction”. In: *Nature methods* 10.6 (2013), pp. 501–507.

[42] Moritz Helmstaedter, Kevin L Briggman, and Winfried Denk. “3D structural imaging of the brain with photons and electrons”. In: *Current Opinion in Neurobiology* 18.6 (2008), pp. 633–641. ISSN: 0959-4388. DOI: DOI:10.1016/j.conb.2009.03.005. URL: http://www.sciencedirect.com/science/article/B6VS3-4W1R9S3-1/2/a103d02b5597e15dc52e612b6f6c5cec.

[43] Moritz Helmstaedter, Kevin L Briggman, and Winfried Denk. “High-accuracy neurite reconstruction for high-throughput neuroanatomy”. In: *Nature neuroscience* 14.8 (2011), pp. 1081–1088.

[44] Moritz Helmstaedter et al. “Connectomic reconstruction of the inner plexiform layer in the mouse retina”. In: *Nature* 500.7461 (2013), pp. 168–174.

[45] Monika R Henzinger and Valerie King. “Randomized fully dynamic graph algorithms with polylogarithmic time per operation”. In: *Journal of the ACM (JACM)* 46.4 (1999), pp. 502–516.

[46] Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. “Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity”. In: *Journal of the ACM (JACM)* 48.4 (2001), pp. 723–760.

[47] Gary B Huang and Viren Jain. “Deep and wide multiscale recursive networks for robust image labeling”. In: *arXiv preprint arXiv:1310.0354* (2013).

[48] Viren Jain et al. “Boundary learning by optimization with topological constraints”. In: *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2010, pp. 2488–2495.

[49] Viren Jain et al. “Learning to agglomerate superpixel hierarchies”. In: *Advances in Neural Information Processing Systems* 2.5 (2011).

[50] Viren Jain et al. “Supervised Learning of Image Restoration with Convolutional Networks”. In: *Computer Vision, IEEE International Conference on* (2007), pp. 1–8. DOI: http://doi.ieeecomputersociety.org/10.1109/ICCV.2007.4408909.

[51] Jeremy Jancsary et al. “Regression Tree Fields—An efficient, non-parametric approach to image labeling problems”. In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE. 2012, pp. 2376–2383.

[52] E. Jurrus et al. "Axon tracking in serial block-face scanning electron microscopy". In: *Medical image analysis* 13.1 (2009), pp. 180–188. ISSN: 1361-8415.

[53] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. "Snakes: Active contour models". In: *International journal of computer vision* 1.4 (1988), pp. 321–331.

[54] Narayanan Kasthuri and Jeff W. Lichtman. *Mouse S1 cortex Automatic Tape-Collecting Ultra Microtome (ATUM)-based Scanning Electron Microscopy (SEM) volume.* `http://www.openconnectomeproject.org`. 2011.

[55] Narayanan Kasthuri et al. "Saturated reconstruction of a volume of neocortex". In: *Cell* 162.3 (2015), pp. 648–661.

[56] Verena Kaynig et al. "Fully automatic stitching and distortion correction of transmission electron microscope images". In: *Journal of structural biology* 171.2 (2010), pp. 163–173.

[57] Michael Kazhdan et al. Color corrected *Mouse S1 cortex Automatic Tape-Collecting Ultra Microtome (ATUM)-based Scanning Electron Microscopy (SEM) volume.* `http://www.openconnectomeproject.org`. 2013.

[58] Michael Kazhdan et al. "Gradient-Domain Processing for Large EM Image Stacks". In: *arXiv preprint arXiv:1310.0041* (2013).

[59] Graham Knott et al. "Serial section scanning electron microscopy of adult brain tissue using focused ion beam milling". In: *The Journal of Neuroscience* 28.12 (2008), pp. 2959–2964.

[60] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems.* 2012, pp. 1106–1114.

[61] Ritwik Kumar, Amelio Vázquez-Reina, and Hanspeter Pfister. "Radon-like features and their application to connectomics". In: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on.* IEEE. 2010, pp. 186–193.

[62] Masaaki Kuwajima, John M Mendenhall, and Kristen M Harris. "Large-volume reconstruction of brain tissue from high-resolution serial section images acquired by SEM-based scanning transmission electron microscopy". In: *Nanoimaging.* Springer, 2013, pp. 253–273.

[63] Haiying Liu, Rama Chellappa, and Azriel Rosenfeld. "Accurate dense optical flow estimation using adaptive structure tensors and a parametric model". In: *Image Processing, IEEE Transactions on* 12.10 (2003), pp. 1170–1180.

[64] Aurélien Lucchi et al. "A fully automated approach to segmentation of irregularly shaped cellular structures in EM images". In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2010.* Springer, 2010, pp. 463–471.

[65] Marina Meilă. “Comparing clusterings—an information based distance”. In: *Journal of Multivariate Analysis* 98.5 (2007), pp. 873–895.

[66] Anish Mittal, Rajiv Soundararajan, and Alan C Bovik. “Making a “completely blind” image quality analyzer”. In: *Signal Processing Letters, IEEE* 20.3 (2013), pp. 209–212.

[67] Rafael Navarro, Justo Arines, and Ricardo Rivera. “Direct and inverse discrete Zernike transform”. In: *Optics express* 17.26 (2009), pp. 24269–24281.

[68] Juan Nunez-Iglesias et al. “Machine Learning of Hierarchical Clustering to Segment 2D and 3D Images”. In: *PloS one* 8.8 (2013), e71715.

[69] N. Pinto et al. “A High-Throughput Screening Approach to Discovering Good Forms of Biologically Inspired Visual Representation”. In: *PLoS Computational Biology* 5.11 (2009). URL: `http://www.rowland.harvard.edu/cox/pdfs/pinto_plos09.pdf`.

[70] William M. Rand. “Objective Criteria for the Evaluation of Clustering Methods”. In: *Journal of the American Statistical Association* 66.336 (1971), pp. 846–850. DOI: `10.1080/01621459.1971.10482356`. eprint: `http://www.tandfonline.com/doi/pdf/10.1080/01621459.1971.10482356`. URL: `http://www.tandfonline.com/doi/abs/10.1080/01621459.1971.10482356`.

[71] Stéphane Ross, Geoffrey J Gordon, and J Andrew Bagnell. “No-regret reductions for imitation learning and structured prediction”. In: *14th International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2011.

[72] Stefan Roth and Michael J Black. “Fields of experts: A framework for learning image priors”. In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 2. IEEE. 2005, pp. 860–867.

[73] Daniel Sage. *Local normalization filter to reduce the effect of non-uniform illumination.* `http://bigwww.epfl.ch/sage/soft/localnormalization/`. Mar. 2011.

[74] R Schalek et al. “ATUM-based SEM for high-speed large-volume biological reconstructions”. In: *Microscopy and Microanalysis* 18.S2 (2012), pp. 572–573.

[75] R Schalek et al. “Development of high-throughput, high-resolution 3D reconstruction of large-volume biological tissue using automated tape collection ultramicrotomy and scanning electron microscopy”. In: *Microscopy and Microanalysis* 17.S2 (2011), pp. 966–967.

[76] Mojtaba Seyedhosseini et al. “Detection of neuron membranes in electron microscopy images using multi-scale context and radon-like features”. In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2011*. Springer, 2011, pp. 670–677.

[77] Kevin Smith, Alan Carleton, and Vincent Lepetit. “Fast ray features for learning irregular shapes”. In: *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE. 2009, pp. 397–404.

[78] Christoph Sommer et al. "ilastik: Interactive learning and segmentation toolkit". In: *Biomedical Imaging: From Nano to Macro, 2011 IEEE International Symposium on*. IEEE. 2011, pp. 230–233.

[79] Shin-ya Takemura et al. "A visual motion detection circuit suggested by Drosophila connectomics". In: *Nature* 500.7461 (2013), pp. 175–181.

[80] Mikkel Thorup. "Near-optimal fully-dynamic graph connectivity". In: *Proceedings of the thirty-second annual ACM symposium on Theory of computing*. ACM. 2000, pp. 343–350.

[81] Srinivas C. Turaga et al. "Convolutional networks can learn to generate affinity graphs for image segmentation". In: *Neural Comput.* 22.2 (2010), pp. 511–538. ISSN: 0899-7667. DOI: `http://dx.doi.org/10.1162/neco.2009.10-08-881`.

[82] Srinivas Turaga et al. "Maximin affinity learning of image segmentation". In: *Advances in Neural Information Processing Systems 22*. Ed. by Y. Bengio et al. Cambridge, MA: MIT Press, 2009, pp. 1865–1873.

[83] Amelio Vazquez-Reina et al. "Segmentation fusion for connectomics". In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, pp. 177–184.

[84] Stefan Wager, Sida Wang, and Percy Liang. "Dropout training as adaptive regularization". In: *Advances in Neural Information Processing Systems*. 2013, pp. 351–359.

[85] Zhou Wang et al. "Image quality assessment: from error visibility to structural similarity". In: *Image Processing, IEEE Transactions on* 13.4 (2004), pp. 600–612.

[86] J. G. White et al. "The Structure of the Nervous System of the Nematode Caenorhabditis elegans". In: *Philosophical Transactions of the Royal Society of London. B, Biological Sciences* 314.1165 (1986), pp. 1–340. DOI: `10.1098/rstb.1986.0056`. eprint: `http://rstb.royalsocietypublishing.org/content/314/1165/1.full.pdf+html`. URL: `http://rstb.royalsocietypublishing.org/content/314/1165/1.abstract`.

[87] Ciyou Zhu et al. "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization". In: *ACM Transactions on Mathematical Software (TOMS)* 23.4 (1997), pp. 550–560.

[88] Aleksandar Zlateski. "A design and implementation of an efficient, parallel watershed algorithm for affinity graphs". PhD thesis. Massachusetts Institute of Technology, 2011.

[89] Aleksandar Zlateski and H. Sebastian Seung. "Image Segmentation by Size-Dependent Single Linkage Clustering of a Watershed Basin Graph". In: *CoRR* abs/1505.00249 (2015). URL: `http://arxiv.org/abs/1505.00249`.

[90] Karel Zuiderveld. "Contrast limited adaptive histograph equalization". In: *Graphic Gems* (1994), pp. 474–485.