UNIVERSITY OF CALIFORNIA SAN DIEGO

Spatial Distribution of Subcellular Organelles in Hippocampal Dendrites from High-Resolution EM Images

A Dissertation submitted in partial satisfaction of the requirements
for the degree Doctor of Philosophy

in

Doctor of Philosophy in Neurosciences with a specialty in Computational Neuroscience

by

Jeffrey Alan Bush

Committee in charge:

Professor Mark H. Ellisman, Chair
Professor Brenda L. Bloodgood
Professor Maryann E. Martone
Professor Terry J. Sejnowski
Professor Larry L. Smarr

2018

Dedication

To my partner and my family for their continual, unwavering, support and encouragement.

In memory of my father for the inspiration and always encouraging me to learn and explore.

Epigraph

All that we are is the result of what we have thought.

Gautama Buddha

Table of Contents

List of Abbreviations

AHE	adaptive histogram equalization
AMIGO-1	amphoterin-induced gene and ORF [Open Reading Frame] 1
AMP	adenosine monophosphate
AP	action potential
ATP	adenosine triphosphate
bAP	back-propagating action potential
BK	large-conductance calcium-activated potassium channel
BLAS	basic linear algebra subprograms
BSE	backscattered electrons
CA1	*Cornu Ammonis* (hippocampus) region 1
CaMK	calcium/calmodulin-dependent protein kinase
CCD	charge-coupled device
CGAL	Computational Geometry Analysis Library
CHM	cascaded/contextual hierarchical model
CICR	calcium-induced calcium release
CLAHE	contrast limited adaptive histogram equalization
EC2	Elastic Compute Cloud
EM	electron microscopy
ER	endoplasmic reticulum
ET	electron tomography
ESEM	environmental scanning electron microscopy
FFT	fast Fourier transform
FIBSEM	focused ion beam scanning electron microscopy
FN	false negatives
FP	false positives

GB	gigabyte (1000^3 or 1 billion bytes)
geoEM	<u>Geo</u>metry Analysis of <u>EM</u> Data
GFP	green fluorescent protein
GHK	Goldman-Hodgkin-Katz
GiB	gibibyte (1024^3 bytes)
GLIA	graph learning library for image analysis
GP	gigapixel (1 billion pixels)
GPU	graphics processing unit
GR	geometric ratio
GUI	graphical user interface
HNS	hierarchical neuronal segmentation
HOG	histogram of oriented gradients
IP3	inositol 1,4,5-trisphosphate
IP3R	IP3 receptor
ITK	Insight Toolkit
JPEG	Joint Photographic Experts Group
KB	kilobyte (1000 bytes)
KChIP	voltage-dependent potassium channel-interacting protein
KiB	kibibyte (1024 bytes)
Kv2.1	potassium voltage-gated channel, Shab-related subfamily, member 1 (aka KCNB1)
Kv4.2	potassium voltage-gated channel, subfamily D, member 2 (aka KCND2)
LDNN	logistic disjunctive normal network
LB	lamellar bodies
LTD	long-term potentiation
LTP	long-term potentiation
mAChRs	muscarinic acetylcholine receptors

mGluRs	metabotropic glutamate receptors
MB	megabyte (1000^2 or 1 million bytes)
MiB	mebibyte (1024^2 bytes)
MP	megapixel (1 million pixels)
MRC	Medical Research Council
MRI	magnetic resonance imaging
NBCR	National Biomedical Computation Resource
NCS	neuronal calcium sensor
NCX	sodium/calcium exchanger
NCMIR	National Center for Microscopy and Imaging Research
NIH	National Institutes of Health
NMDA	N-methyl-D-aspartate
NSF	National Science Foundation
PKA	protein kinase A
PNG	portable network graphics
px	pixel
PyCHM	Python version of CHM
RAM	random access memory
RSS	resident set size
RyR	ryanodine receptor
S3	Simple Storage Service
SA	spine apparatus
SBFSEM	serial blockface scanning electron microscopy
SCI	Scientific Computing and Imaging Institute
SEM	scanning electron microscopy
SERCA	sarcoendoplasmic reticulum calcium ATPase

SIFT	scale-invariant feature transform
SK	small-conductance calcium-activated potassium channel
SLASH	Scalable system for Large data Analysis and Segmentation utilizing a Hybrid approach
SP	synaptopodin
SS	subsampling
SSC	subsurface cisternae
SSD	solid state drive
ssTEM	serial section transmission electron microscopy
STDP	spike-timing-dependent plasticity
SVR	surface to volume ratio
TB	terabyte (1000^4 or 1 trillion bytes)
TEM	transmission electron microscopy
TiB	tebibyte (1024^4 bytes)
TIFF	tagged image file format
TN	true negatives
TP	true positives
TP	terapixel (1 trillion pixels)
UCSD	University of California San Diego
VGCC	voltage-gated calcium channel

List of Figures

List of Tables

Acknowledgements

This project would not have been possible without the amazing support of the entire NCMIR lab and the SLASH group. From the beginning, the entire lab has contributed to my success. Early on many members of the NCMIR lab provided training so that I could collect my own samples. Mason Mackey in particular helped me learn about the diverse set of EM equipment in the lab. The final datasets for this project were prepared and collected by Tom Deernick – an absolute master of the art of EM sample preparation and collection. Moving into computational tasks, the NCMIR IT team, including Edmond Negado and Vicky Rowley, provided invaluable assistance in getting computational resources and data in the right places. Chris Churas introduced me to the near-endless supplies of CPU-years with the numerous supercomputers we have access to and was always there to discuss any part of my project. Mojtaba Seyedhosseini and Mehran Javanmardi from the SCI group at the University of Utah provided the technical support for understanding the mathematics within the CHM segmentation algorithm while Alex Perez at NCMIR provided the insight of how to use it for my data. Ting Liu from the SCI group frequently answered questions about the HNS/GLIA cell segmentation algorithms built on top of CHM. Many others in the lab discussed and analyzed the various computational approaches, including David Lee and Niko Komin. Andrew Noske taught me how to create my own manual segmentations and helped with the integration of the Livewire assisted segmentation program into IMOD so that everyone in the lab, and electron microscopists around the world, could use it to accelerate the process of manually tracing their data.

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562. This project was supported by grants from the National Center for Research Resources (5P41RR008605-19) and the National Institute of General Medical Sciences (P41 GM103426) from the National Institutes of Health (NIH). Additionally, this project utilized an allocation of computational resources from the NIH Commons Credit Pilot Award (CCREQ-2017-03-00017).

Vita

2006-2009	Research Assistant, Computational Biology, Rensselaer Polytechnic Institute
2007-2009	Teaching Assistant, Computer Science, Rensselaer Polytechnic Institute
2009	Bachelor of Science, Bioinformatics/Molecular Biology, Rensselaer Polytechnic Institute
2009	Bachelor of Science, Biochemistry/Biophysics, Rensselaer Polytechnic Institute
2009	Bachelor of Science, Computer Science, Rensselaer Polytechnic Institute
2010-2011	Teaching Assistant, Neurodynamics, University of California San Diego
2012	Master of Science, Neurosciences, University of California San Diego
2012-2015	Graduate Student Researcher, University of California San Diego
2014	Qualcomm Engineering Intern, San Diego
2015	iD Tech Academies Instructor, Palo Alto
2016-2017	Visiting Instructor, Computer Science, Ursinus College
2017-2018	Instructor, Computer Science, Moravian College
2018	Doctor of Philosophy, Neurosciences, University of California San Diego

ABSTRACT OF THE DISSERTATION

Spatial Distribution of Subcellular Organelles in
Hippocampal Dendrites from High-Resolution EM Images

by

Jeffrey Alan Bush

Doctor of Philosophy in Neurosciences with a specialty in Computational Neuroscience

University of California San Diego, 2018

Professor Mark H. Ellisman, Chair

Neurons use several methods to integrate incoming information to make a decision about whether to activate an action potential and send information on to other neurons. This process adapts over time and provides the neurons with the ability to learn. One of the forms of learning that is not well understood is branch point plasticity, the ability for different branches in the dendritic arbor to change their ability to conduct information over time. In this project tools are developed to study the structural changes of branch points and indirectly the chemical changes at branch points to better understand the underlying mechanisms behind branch point plasticity. To this end large quantities of neuropil data need to be analyzed. Modern electron microscopy techniques can provide massive quantities of biological image data at extremely high magnification, but the ability to process this data and obtain usable information is a major bottleneck.

Machine learning tools were used to automatically segment cell membranes and mitochondria in large volumes of neuropil electron microscopy data. The new implementation of the algorithms improved their accuracy and efficiency for the given datasets. The algorithms now provide near-human accuracy for organelle and membrane detection at the same speed that data can be acquired from the microscopes using off-the-shelf desktop machines making this tool accessible to labs without specialized computing resources. Specialized programs were then developed to analyze the geometry of the branch points along with the spatial distributions of mitochondria and endoplasmic reticulum relative to the branch points. Using these tools on a sample set of data shows that mitochondria and endoplasmic reticulum volume percentages fluctuate with the distance from the soma. Moving forward, these tools can be used to analyze large datasets to discover the underlying mechanisms of branch point plasticity.

Introduction

In 2013 the Human Genome Project declared success after mapping almost the entire human genome and made this data available to the entire scientific community (Schmutz, et al., 2004), providing numerous medical and scientific benefits along with a much deeper understanding of biology (Hood & Rowen, 2013). The success of the project was an inspiration to map all the neural connections in the brain to better understand its mechanisms. In 2005 the terms connectome – the comprehensive map of all of the neural connections in the brain (Sporns, Tononi, & Kötter) and connectomics – the study and production of connectomes (Hagmann) were coined, modeled after the terms genome and genomics. This work was born out of hodology – the study of the pathways in the brain and the interconnections of its cells – which has been around for centuries (Dejerine & Dejerine-Klumpke, 1895; Van, 1949). In 2009 the NIH launched the Human Connectome Project (2009) to push forward the mapping of the neuronal connections in the human brain. In 2013 the United States began the BRAIN Initiative (Insel, Landis, & Collins) and the European Union began the Human Brain Project (Abbott A.) both of which are part of a world focus on mapping the connectome. For the most part efforts have been focused on completing the connectome of various model organisms, including *C. elegans* (White, Southgate, Thomson, & Brenner, 1986; Chen, Hall, & Chklovskii, 2006), *D. melanogaster* (Chklovskii, Vitaladevuni, & Scheffer, 2010), mice (Bock, et al., 2011), and humans. However, connectomics is not the end-all solution to understanding the brain, just as knowing the sequence of the entire human genome was not the end of the road, it is simply a single step of the path. For the human genome, we may know the sequence of the genome, but we do not know what it all means, as revealed by the fact that only around 2% of the human genome encodes protein sequences and the related C-enigma (Bennett & Leitch, 2005; Gregory, 2005). Similarly, the ability to have the map of all connections in the brain will not be the end of the road. We need to also figure out how single neurons function and the computations that they can perform.

The focus of this research is to create new computational tools to study the subcellular organization of neurons. In particular, applying these new tools to the distribution of the endoplasmic reticulum (ER) within the dendrites of neurons and how that distribution relates to the branching patterns of the dendritic tree. This will provide tools to study the intricacies of individual neurons and offer some possible explanations for some of the additional complexities that occur within them.

1 Dendritic Signal Integration in Neurons

To understand how neurons integrate all of the synaptic inputs along their dendrites to make a "decision" on whether or not to pass a signal onto other cells the basics need to be established. This section covers the basics of neurons and how they are modeled and then goes into several details of information integration within the dendrites. Included details are: the role that the endoplasmic reticulum (ER) plays, the various plasma membrane potassium channels that have roles in this process, how the morphology of the dendrites influences the integration particularly at branch points, and how all of these features work together to perform dendritic signal integration. Dendritic integration and compartmentalization is believed to play a direct role in spatial information processing (Morita, 2008; Nolan, et al., 2004). Additionally, a rich set of biological mechanisms can dynamically change the dendritic computation in response to cell activity (Johnston & Narayanan, 2008).

1.1 Neuron Modeling

This section covers the basics of electrical properties of neurons and how they are modeled. Additionally, some of the complexities of neurons which are not captured in these models are discussed along with the underlying biochemical mechanism behind these dynamic behaviors: protein phosphorylation.

1.1.1 Integrate-and-Fire Models of Neurons

Neurons receive inputs from other neurons through synapses on their dendrites. The activation of the synapses causes an electrical current to flow across the cell membrane, the barrier between the intracellular and extracellular material, which changes the voltage across the cell membrane. If enough synapses are activated at the same time, then the voltage change will surpass a threshold voltage and cause the neuron to send an action potential, a large electrical signal, down its axon which in turn causes the activation of all synapses along the axon which are then the inputs to the next set of neurons on their dendrites. After an action potential the voltage across the membrane returns to its “resting” state. This resetting process takes some time during which another action potential cannot be fired, this period is called the refractory period.

This basic concept of a neuron is the basis for the integrate-and-fire model of a neuron. In 1907 Louis Lapicque developed one of the first models of a neuron based on this simple electric principle (Abbott L. F., 1999). This model simply takes the fact that the charge in a neuron builds up over time (integrates) until a threshold is reached and then the neuron creates an action potential (fires), and resets to its resting voltage potential. The original model was a simple linear differential equation representing the electrical circuit consisting of a parallel resistor and capacitor followed by a non-linear handling of the threshold voltage and resetting back to rest. This simple model is very easy to compute and can accurately model many prototypical neuron behaviors (Koch & Segev, 1999). Due to its simplicity and easy of computation, it is now the basis for the artificial “neurons” used in artificial neural networks (Abbott L. F., 1999).

However, this model has many shortcomings. The most immediately observable one is that it does not implement the concept of a refractory period. For simplicity this can be implemented as another nonlinearity that simply clamps the membrane voltage for a period after an action potential so that the voltage across the membrane cannot change until the refractory period is over. Over the years additional enhancements have been made to the integrate-and-fire model for specific purposes. Some examples are the exponential integrate-and-fire model which incorporates additional information about the sodium channels (Fourcaud-Trocmé, Hansel, van Vreeswijk, & Brunel, 2003), the Izhikevich model which aims to be more biophysically realistic while maintaining high computational efficiency (Izhikevich, 2003), and fractional-order leaky integrate-and-fire models which incorporate the fact that some neurons perform fractional differentiation (Teka, Marinov, & Santamaria, 2014; Lundstrom, Higgs, Spain, & Fairhall, 2008).

Since the brain consists of around 100 billion neurons (Braitenberg, 2001), the modeling of the entire brain would currently necessitate the use of a simple model like integrate-and-fire or one of its variations to simulate each neuron at reasonable speeds. Even then, only extremely powerful hardware would be able to run the simulation and we ever need to know all the connections between the neurons.

However, all these models only pick up on specific behaviors of specific neurons. At their core they all assume a neuron is a simple linear integration of its inputs followed by a nonlinear thresholding. They end up modeling the firing patterns of a neuron instead of the biological events occurring within the system and incidentally only model the prototypical patterns seen for neurons and not the actual wide range of behaviors that neurons exhibit.

1.1.2 Electrochemical Models of Neurons

The electrical currents across the plasma membrane are carried by ions and thus to incorporate additional biophysically realistic complexities, information about them must be added into the neuronal models.

The plasma membrane is semipermeable, selectively allowing some ions across, into or out of the cell, while blocking others. Additionally, each of the ions that can cross the membrane do so at a specific rate. Neurons strictly regulate the ion flow across the plasma membrane by adding ion channels, complex membrane-bound proteins that form a hole in the membrane of a specific size to allow only certain ions across. Some ion channels are always open, allowing a constant “leak” of ions across the membrane, while other ion channels only open when the voltage across the membrane reaches a certain threshold or if certain ligands are present. In general, ions prefer to move down their electrochemical gradient, from the side of the membrane which has a higher concentration of the ion and charge to the side with lower concentration and charge. If this was allowed to continue for too long then an equilibrium would be reached with the charge and concentrations remaining the same on both sides of the membranes, resulting in no overall current or voltage potential across the membrane. To overcome this, neurons also incorporate ion pumps into the membrane which constantly spend energy to move ions across the membrane against the electrochemical gradient. Thus, the “resting state” for a neuron actually requires energy to be maintained. This system allows a low current, and thus voltage potential, at rest for the neuron to be ready to quickly change voltage potential as necessary.

Each ion has its own reversal potential, or voltage potential across the membrane. The reversal potential is different for every ion and is dependent on the electrical charge of the ion and the concentration of the ion on both sides of the membrane. The reversal potential for each ion is critical in determining the overall voltage across the membrane. They are calculated using the Nernst equation:

$$V_{ion} = \frac{V_t}{z_{ion}} \ln\left(\frac{[\text{ion}]_{out}}{[\text{ion}]_{in}}\right)$$

where V_{ion} is the reversal potential for a specific ion, z_{ion} is the charge of the ion, $[\text{ion}]_{out}$ and $[\text{ion}]_{in}$ are the extracellular and intracellular ion concentrations respectively, and V_t is the thermal voltage constant ($V_t = {}^{RT}/_{F} \cong 26.7\text{ mV}$ for mammals). The voltage potential for each ion can be calculated individually using this equation along with the known charge values and the relatively easy measurement of the concentrations of the ions. The concentrations for the most prominent ions in a typical mammalian neuron at rest are given in Table 1.1 along with the calculation of their reversal potentials. It can be seen

Table 1.1. Concentrations and reversal potentials for the most prominent ions in a typical mammalian neuron at rest. From (Electrical Signals of Nerve Cells, 2001).

Ion	Ion Charge (z)	Ion Concentration (mM)		Reversal Potential (mV)
		Intracellular	Extracellular	
Potassium (K^+)	$+1$	140	5	-89
Sodium (Na^+)	$+1$	5 to 15	145	61 to 90
Chloride (Cl^-)	-1	4 to 30	110	-35 to -89
Calcium (Ca^{2+})	$+2$	0.0001	1 to 2	123 to 132

that some ions, like potassium and chloride, create a negative voltage potential across the membrane while others, such as sodium and calcium, create a positive voltage potential across the membrane. The charge of the ion and which side of the membrane has the higher ion concentration determines the sign of the reversal potential.

However, none of these ions act alone and they must be considered collectively to determine the total voltage potential across the membrane. David Goldman, Alan Hodgkin, and Barnard Katz developed what is now called the Goldman-Hodgkin-Katz (GHK) voltage equation (Hodgkin & Katz, 1949; Goldman, 1943). The equation that combines potassium, sodium, and chloride ions is as follows:

$$V_m = V_t \ln\left(\frac{P_K[\mathrm{K}^+]_{out} + P_{Na}[\mathrm{Na}^+]_{out} + P_{Cl}[\mathrm{Cl}^-]_{in}}{P_K[\mathrm{K}^+]_{in} + P_{Na}[\mathrm{Na}^+]_{in} + P_{Cl}[\mathrm{Cl}^-]_{out}}\right)$$

where V_m is the voltage potential across the membrane and P_{ion} is the relative permeability of the membrane to that ion. This is essentially a weighted sum of the Nernst equation for each ion with each being weighted depending on its relative permeability. An ion with a higher permeability, and thus great ease in crossing the plasma membrane, has a greater influence on the resting membrane potential since it has more movement into or out of the cell and thus generates a larger proportion of the current across the membrane.

The chloride concentrations appear to be "upside down" from the other ion contractions in the equations because it has a charge of -1 while the other ions all have charges of $+1$. Additionally, calcium is not included since it has a charge of $+2$ and incorporating it makes the equation significantly more complex (Spangler, 1972). Since the concentration of calcium is relatively low compared to the other ions and has a very low permeability ($\ll 0.1\%$ of potassium) in the resting state, it does not play a significant role in the resting membrane voltage potential.

Of the remaining common ions, potassium has the highest permeability at rest. If we define $P_K = 1$ and have all the other ion permeabilities relative to it this gives us the approximate permeabilities of $P_{Cl} = 0.45$ and $P_{Na} = 0.04$ at rest. Since potassium dominates in the GHK equation, the resting voltage potential across the membrane will be relatively close to its reversal potential. For the example data given, the membrane potential is approximately $V_m = -72\ \mathrm{mV}$ which is much closer to $V_K = -89\ \mathrm{mV}$ than $V_{Na} = 90\ \mathrm{mV}$.

So far this has all described a neuron "at rest" which means the steady-state the neuron will achieve if left without stimulation for a long enough period of time. In response to specific stimuli the ionic permeabilities of the membrane undergo dramatic changes. The most common example is the action potential which is typically thought of as a massive change in voltage potential from negative to positive and back down to negative again. If we consider it in more detail, we can see that what is really changing is the membrane permeability to specific ions. When the membrane voltage potential shifts due to a stimulus from its resting value around $-70\ \mathrm{mV}$ to a threshold voltage around $-55\ \mathrm{mV}$ which triggers a massive shift in the permeability of sodium, which in turn causes the membrane voltage potential to prefer to be closer to the reversal potential of sodium instead of potassium and it begins to rapidly change reaching a peak around $+40\ \mathrm{mV}$ before the relative permeability of sodium begins decreasing as potassium ion's permeability increases. This causes the membrane voltage potential to then once again favor potassium's reversal potential instead of sodium's and thus quickly drops down all the way to potassium's reversal potential around $-90\ \mathrm{mV}$ and then gradually resets moving back towards the resting potential. These significant changes in the membrane potential are caused by changes in the permeability of the plasma membrane to specific ions, constantly shifting the preferred voltage across the membrane towards one of the ion's reversal potentials. At the physical level, the changes in permeability are carried out by various ion channels opening and closing in response to stimuli such as a certain membrane voltage or the presence of a specific chemical.

This dynamic nature was described by Alan Hodgkin and Andrew Huxley (1952) in their seminal work for which they earned the Nobel Prize in 1963. They used the action potentials in the squid giant axon as a model. Like Lapicque, who created the integrate-and-fire model, they created a model based on an electrical circuit. Once again, the lipid bilayer of the plasma membrane is represented as a capacitor and a resistor in parallel is used to represent the leak current. However, that is where the integrate-and-fire model ends. Hodgkin and Huxley added variable resistors in parallel for each of the different ion channels. Each of those resistors is tied to a battery that represents the electrochemical gradient for that ion. Using this model a differential equation can by derived the describes the changing nature of the membrane:

$$I = C_m \frac{\mathrm{d}V_m}{\mathrm{d}t} \qquad\qquad I = \sum I_{ion} \qquad\qquad I_{ion} = g_{ion}(V_m - V_{ion})$$

where C_m is the capacitance per unit area of the membrane (commonly $1\mu\mathrm{F}/\mathrm{cm}^2$), $\mathrm{d}V_m/\mathrm{d}t$ represents the voltage potential change across the membrane over time, I is a sum of all the different ionic currents across the membrane per unit area, I_{ion} is a specific ionic current per unit area, and g_{ion} is the conductance (inverse of resistance) per unit area of that ion across the membrane. This formula is a linear approximation of the GHK current density equation (not shown) which is related to the above GHK voltage equation. In this formulation, the amount of current generated by one ion species is based on how far the membrane potential is from that ion's reversal potential and its conductance. At rest, the membrane potential is quite far from the reversal potential of sodium, which means that $V_m - V_{Na}$ is a relatively large value. However, we know that there is not a large sodium current at rest and thus the conductance for sodium at rest must be quite low. The conductance of an ion is correlated with its permeability, which goes along with this since we know that the permeability of sodium is very low at rest.

For the model to be dynamic the permeability of the membrane needs to change over time as well, thus the conductances for the ions cannot be constants but instead based on differential equations as well. For example, the potassium and sodium conductances described by Hodgkin and Huxley changed based on the membrane voltage and are represented as:

$$g_K = \overline{g}_K n^4 \qquad g_{Na} = \overline{g}_{Na} m^3 h$$

$$\frac{dn}{dt} = \alpha_n(V_m)(1-n) - \beta_n(V_m)n \qquad \frac{dm}{dt} = \alpha_m(V_m)(1-m) - \beta_m(V_m)m$$

$$\frac{dh}{dt} = \alpha_h(V_m)(1-h) - \beta_h(V_m)h$$

where $\overline{g}_K$ and $\overline{g}_{Na}$ are the maximum conductances of the potassium and sodium, n, m, and h are dimensionless quantities that range from 0 to 1, and $\alpha_p(V_m)$ and $\beta_p(V_m)$ are rate constant functions. They also described a leak conductance that was constant so $g_{leak} = \overline{g}_{leak}$.

The state variables n, m, and h have physiological correlates (which were unknown to Hodgkin and Huxley at the time but have been discovered since). The potassium channel has 4 subunits that must all be activated simultaneously for it to open and so that potassium ions can pass through. The n state value represents the activation of a single subunit while n^4 represents all four subunits activated and thus the channel opening. The sodium channel has 3 subunits that must be activated (m^3) and one subunit that must be inactivated (h) simultaneously to open and allow sodium ions to pass through. The fact that h is an "inactivation probability" instead of an activated probability is dictated by the choice of the rate constant functions α_h and β_h and is otherwise not treated any differently.

The rate constant functions are usually based on Boltzmann equations which describe how the activation of those subunits behave if the membrane was held at a fixed voltage of the current membrane voltage potential. The original Hodgkin and Huxley paper gives the functions as follows:

$$\alpha_n(V_m) = \frac{0.01(10-V_m)}{\exp\left(\frac{10-V_m}{10}\right)-1} \qquad \alpha_m(V_m) = \frac{0.1(25-V_m)}{\exp\left(\frac{25-V_m}{10}\right)-1} \qquad \alpha_n(V_m) = 0.07\exp\left(-\frac{V_m}{20}\right)$$

$$\beta_n(V_m) = 0.125\exp\left(-\frac{V_m}{80}\right) \qquad \beta_n(V_m) = 4\exp\left(-\frac{V_m}{18}\right) \qquad \beta_h(V_m) = \frac{1}{\exp\left(\frac{30-V_m}{10}\right)+1}$$

Notice how that the α_h and β_h functions appear switched compared to the other rate constant functions due to the fact that h represents an inactivation subunit. All the constants in these functions, and the form of the functions themselves, were derived experimentally from the giant axon of the squid by Hodgkin and Huxley[1].

The final values that are needed to be able to simulate the membrane potential over time are the maximum conductance values $\overline{g}_K$, $\overline{g}_{Na}$, and $\overline{g}_{leak}$. These values are directly related to the maximal permeability of the ions, i.e. when all of the ion channels are in the full-open state. The values used by Hodgkin and Huxley in units of $\mathrm{mS/cm^2}$ or equivalently $1/(\mathrm{m\Omega \cdot cm^2})$ are:

$$\overline{g}_K = 36 \qquad \overline{g}_{Na} = 120 \qquad \overline{g}_{leak} = 0.3$$

Unlike integrate-and-fire model neurons, Hodgkin-Huxley model neurons do not need any special rules to simulate an action potential or refractory periods, all of that is encoded in the rate constant functions for each of the channels just as it is in real neurons.

While we can now describe the voltage potential changing at a single point on the plasma membrane, frequently this is not sufficient for complete neurons as it takes time for the signal to propagate along the membrane. The previous model needs to be combined with the cable theory to accomplish this. Cable theory was originally developed to model signal transmission in underwater telegraph cables and derived from the ubiquitous 1D heat equation (Thomson, 1855) but in the mid-twentieth century it was recognized as an important part of the electrical transmission of signals in neurons (Hodgkin & Rushton, 1946; de Nó, 1947). It functions by breaking up the entire membrane into smaller compartments. Each membrane compartment has its own Hodgkin-Huxley model and each compartment lets current flow to neighboring compartments with a longitudinal resistance (R_l). If each of the compartments is small enough and the Hodgkin-Huxley model for them is detailed enough this can

[1] Some minor changes have been made. In the original paper the voltage reference was reversed and thus their voltage measurements were negative from the modern-day standard.

produce very biophysically realistic models of true neurons, including many intricacies within the dendrites.

The traveling dynamics of action potentials along the membrane can be studied with this model. The actual electrical signal does not simply move along the membrane. Instead, an actual potential occurs at a single location, causing a change in voltage potential and change in ion concentration. Those ions diffuse within the cytosol and extracellular space of the cell. At a neighboring location, if the amount of voltage potential caused by the diffusion of the ions causes the membrane to reach the threshold, then another spike is created at that location. If the sites are too far apart at the threshold or the original spike is not large enough, then the diffusion of the ions may not cause a large enough increase further down the membrane and the action potential will not propagate. This is called regenerative propagation because at every location the spike must be regenerated. Without this regeneration process, action potentials would not travel large distances at all and only nearby locations would sense a slight increase in the voltage potential across the membrane.

While a highly detailed multi-compartment model can provide many qualities of biological neurons, it can be very computationally intensive to simulate, needing to potentially evaluate thousands of differential equations for each time step for a single neuron. This does not scale well for several neurons let alone the approximately 100 billion neurons in the entire human brain. Several variations of the Hodgkin-Huxley model have been made in an attempt to make it more computationally feasible while keeping some of the biophysical realistic properties, which include the FitzHugh-Nagumo model which only requires two derivatives per neuron but no longer has a clear derivation from biology (FitzHugh, 1961; Nagumo, Arimoto, & Yoshizawa, 1962) and the Morris-Lecar model which combines calcium and potassium channels by combining the Hodgkin-Huxley and FitzHugh-Nagumo models (Morris & Lecar, 1981).

Ultimately, multicompartment Hodgkin-Huxley models represent a high level of biological detail but can be computationally expensive for highly detailed models involving large number of compartments or neurons. On the other hand, integrate-and-fire neurons can represent many of the prototypical behaviors of neurons with small amount of computational effort required, allowing them to be scaled to numerous interconnected neurons in a single simulation.

1.1.3 Neuronal Complexity Not Captured with Traditional Models

Neurons have several additional nonlinear behaviors beyond thresholding and refractory period that have been identified including: synaptic plasticity, synaptic noise, differences in somatic and dendritic conductances, distribution of synapses, and morphology of the cell (Silver, 2010; Ramaswamy & Markram, 2015).

The most commonly studied behavior beyond the simple integrate-and-fire model is spike-timing-dependent plasticity (STDP) which results from temporal coincidences that occur between synaptic activation and action potentials (APs). STDP was first theorized to exist in 1973 (Taylor) and since then it has been shown that when an AP is sent down the axon of a neuron an additional signal is sent into the dendritic tree called a back-propagating action potential (bAP). If the bAP arrives at the postsynaptic structure within a small window of time after the synapse was activated, then the strength of the synapse is increased so that the next time the synapse is activated it is more likely to cause the neuron to fire. If the bAP arrives outside of the window, then the strength of the synapse is weakened. This mechanism is believed to be the one of the main underlying machineries of learning within the brain (Feldman, 2012).

There are integrate-and-fire (Tavanaei & Maida, 2017) and compartmental models (Wilmes, Schleimer, & Schreiber, 2017; Frady, 2009) that include bAPs and STDP. However, STDP has numerous intricacies that are usually glossed over when modeled. An example of this is that many neurons have highly complex dendritic arborizations and not all branches within the dendrites are equally penetrated by the bAP and thus some synapses can only perform this coincidence detection under special circumstances such as when there are several action potentials in succession (Feldman, 2012).

While synaptic plasticity, and more specifically STDP, has been shown to allow neurons to change the weighting of individual inputs (Feldman, 2012), they still fundamentally represent neurons that are simple integrate-and-fire models with an additional method to change over time. Neurons have been shown to be able to perform addition, multiplication (Silver, 2010), and fractional differentiation (Lundstrom, Higgs, Spain, & Fairhall, 2008) of their signals. This wide spread of behaviors cannot be explained with this simple model, even a model that can "learn", and require additional forms of non-linearities. Even though several of these have been studied, and some in great depth, no complete system has been created. The more that is learned about individual neurons, the more it becomes apparent that each neuron has significant processing power on its own (Silver, 2010; Ujfalussy, Makara, Branco, & Lengyel, 2015). In fact, Almog and Korngreen argue that not only are simple integrate-and-fire neuron models insufficient but even more advanced compartmental models are still quite lacking and we must continue modeling more complex neurons (2016). There is still much to learn about the coding properties of individual neurons and how the behavior of neurons changes over time.

Most of these neuronal complexities are implemented by changing the behavior of the proteins that make up the ion channels that conduct the electrical signals. Many proteins do not have a fixed behavior for their lifetime and with minor post-translational modifications[2] can exhibit alternate behaviors. Without this flexibility the proteins would have to be broken down and replaced with new proteins every time a minor change was needed, wasting time and energy. Many of the modulations that occur to proteins in neurons must happen at time scales that are not possible with the recycling of entire proteins. Instead, proteins can be modulated over time to tweak their behavior. For voltage-gated ion channels, a common change is to make the channel more or less sensitive to voltage across the plasma

[2] The modifications that occur after the protein structure has been translated from the RNA sequence.

membrane and thus changing things like the threshold voltage or action potential peak or the membrane. Another modification that occurs is the distribution of the ion channels with some modifications causing the ion channels to cluster into tight groups while others make it more evenly distributed and spread out (Levitan, 1994).

One of the most common post-translational modifications that is both reversible and dynamic is phosphorylation and it is estimated that 30% of all cellular proteins are targets of phosphorylation (Cohen, 2000). Many proteins have several phosphorylation sites, with each one possibly having a different set of proteins regulating whether it is phosphorylated and the set of resulting behaviors. Protein kinases are enzymes that add a phosphate group to another protein at a specific site, causing it to be phosphorylated. Some common protein kinases are CaMKI and CaMKII (Calcium/Calmodulin-dependent protein kinase I and II) which activate under increased calcium concentration (Yamauchi, 2005). Phosphatases are enzymes that remove a phosphate group from another protein at a specific site, such as the common calcineurin which is also calcium/calmodulin-dependent (Klee, H, & Krinks, 1979).

1.2 The Endoplasmic Reticulum and Cable-In-Cable Theory

Each independent signal being transmitted through the dendrites will need a different mode of transmission. Different signals already discussed are the synapses being activated causing the soma to generate an action potential down the axon and a bAP that travels up the dendrites back towards the synapses. For high-speed communication some propagating electrochemical gradient will be needed across a membrane. Thus, to have different modes of transmission the cell needs to either use a different ion across the membrane already being utilized or to use a separate membrane system that is not connected to the plasma membrane. Synapse-to-soma and bAP signals utilize the first technique and are both carried across the plasma membrane but primarily use sodium and calcium respectively for the depolarization. Both use potassium for repolarization, but each uses potassium in unique ways as to reduce interference between the signals. However, other independent signals have been identified in dendrites and no other ions seem to play predominant roles. Thus, they must use another membrane to carry the signals.

Another membrane system within the dendrites is the smooth endoplasmic reticulum (SER, simply referred to as ER in this paper). It has been shown that calcium signals along the ER within the dendrites may play a role in many different behaviors of a neuron, from using synaptic input to regulate genomic expression (Hagenston & Bading, Calcium signaling in synapse-to-nucleus communication, 2011) to influencing the propagation of calcium waves (Neymotin, et al., 2015) and soliton waves (Poznanski, et al., 2017).

Since ER enables a mostly independent electrochemical signal within the dendrites, the cable-in-cable hypothesis (Shemer, Brinne, Tegnér, & Grillner, 2008) proposes that neurons do not just operate as a single cable with the voltage potential across the plasma membrane of the cell but instead have two nested cables, with the inner cable being the ER which uses the calcium as the voltage carrier across its membrane and the outer cable being the plasma membrane which primarily uses sodium, calcium, and potassium. The smooth ER forms a contiguous mass in the dendritic tree acting as a calcium tunnel (Martone, Zhang, Simpliciano, Carragher, & Ellisman, 1993; Terasaki, Slater, Fein, Schmidek, & Reese, 1994; Petersen, Tepikin, & Park, 2001), protruding even into very thin compartments such as presynaptic spine heads (Wu, et al., 2017). This allows for two mostly independent, fast propagating electrochemical signals to be conducted throughout the dendrites. They are not completely independent as they share the cytosol with each other and thus have the potential to influence the voltage potential or ionic concentrations used by each other (Shemer, Brinne, Tegnér, & Grillner, 2008). In some regions, where the ER is close to the plasma membrane, or when it takes up a significant portion of the cytosolic volume such as in the necks of dendritic spines, this effect becomes very prominent (Kuwajima, Spacek, & Harris, 2012). Additionally, the plasma membrane contains calcium ion channels, such as various voltage-gated calcium channels (VGCCs) and NMDA receptors on presynaptic terminals on spine heads which means that intracellular calcium can influence and be influenced by both the current across the plasma membrane and the ER membrane (Simms & Zamponi, 2014).

Since the current carrier across the ER membrane is calcium, the concentration of calcium on the outside and inside of the cell, along with the availability of calcium within the ER influences its behavior. The typical resting state of a neuron has very low free cytosolic calcium, around 100 nM (Simms & Zamponi, 2014), while extracellular calcium concentration is around ten thousand times that at about 1 mM (Lu, et al., 2010). This low cytosolic level is maintained by efficient sarcoendoplasmic reticulum calcium ATPase (SERCA) pumps, sodium/calcium exchangers (NCX), and cytosolic buffers such as calbindin (Ross, 2012). However, within a cell there are large stores of calcium ions within the ER or bound to calcium buffers. It is difficult to quantify exactly how much calcium is stored within the ER, but estimates place it around that of the extracellular space or lower, but orders of magnitude higher than the cytosolic free calcium concentration in the cytosol (Bygrave & Benedetti, 1996). The more important metric is how much calcium can be readily released from the ER into the cytosol. The ER in layer 5 rat neocortical pyramidal neurons can cause local calcium concentrations to reach 5 μm (Larkum, Watanabe, Nakamura, Lasser-Ross, & Ross, 2003) which is significantly higher than the calcium concentration increases along the plasma membrane from VGCCs during bAPs (Maravall, Mainen, Sabatini, & Svoboda, 2000).

Simulations of the cable-in-cable model has proved challenging to create in realistic ways since direct measurements of activity along the ER are difficult and simulating them may require spatially realistic simulation systems, such M-Cell or STEPS, which require significantly more computational resources then the traditional multi-compartmental single-cable simulation environments such as NEURON and GENESIS (Anwar, et al., 2014). Even with these issues, attempts have been made using simplified models and estimations of parameters (Shemer, Brinne, Tegnér, & Grillner, 2008; Neymotin, et al., 2015). These models were able to produce many of the studied phenomenon and make predictions about which underlying reactions must be taking place.

1.2.1 Calcium Waves, Calcium Sparks, and Calcium-Induced Calcium Release

Collectively these observations of the dendritic ER indicate that one of its major roles is to store calcium internally and release it to perform intracellular signaling. In addition to bAPs, which travel along the plasma membrane and use calcium, there are two other major calcium events in neurons, both of which utilize the ER: calcium waves and calcium sparks (Ross, 2012). A calcium wave is a widespread event that starts in a particular location and spreads through the dendrites at a speed of approximately 100 µm/s (Nakamura, Lasser-Ross, Nakamura, & Ross, 2002). The plasma membrane based bAPs which also use calcium propagate significantly faster, on the order of 1000x faster (Stuart & Sakmann, 1994). Calcium sparks are much more localized events with very minimal spread, fast to reach their peak, and stochastic. They were first observed in cardiac myocytes (Cheng, Lederer, & Cannell, 1993) and *X. laevis* oocytes (Parker & Ivorra, 1990) where they were named calcium sparks and calcium puffs respectively (Cheng & Lederer, 2008). It was several years before similar events were observed in neurons that have features of both the calcium sparks and puffs identified earlier. These are typically called sparks in neurons (Koizumi, et al., 1999).

The cytosolic calcium level in both calcium waves and calcium sparks raises slightly causing the ER to release additional calcium. In response to the now higher levels of cytosolic calcium even more calcium is released from the internal ER stores. This repeats until either the calcium stored in the ER is deleted or some other molecular signal prevents the release of calcium. This phenomenon is termed calcium-induced calcium release (CICR). Both of the primary ion channels used by the ER, inositol 1,4,5-trisphosphate receptors (IP3Rs) and ryanodine receptors (RyRs) channels, release calcium in response to raised cytosolic calcium (Ross, 2012).

Calcium waves are primarily generated by IP3Rs. IP3Rs also require IP3 to be activated which means that they only open in the presence of both IP3 and calcium ions (Iino & Endo, 1992). At resting levels of cytosolic free calcium, high levels of cytosolic IP3 are required to activate the channel (Foskett, White, Cheung, & Mak, 2007). At elevated levels of cytosolic free calcium less IP3 is required for activation. However, at resting levels of cytosolic IP3, no amount of free calcium can activate the receptors. This means that the first set of IP3Rs can be activated just with high levels of IP3. This causes the channel to release free calcium into the cytosol giving neighboring IP3Rs the ability to open with lower amounts of IP3. This regenerative and propagating calcium release continues until the wave reaches a region where the cytosolic level of IP3 is lower than necessary to activate the receptors (Ross, 2012). One of the main ways that IP3 levels in the cytosol are raised is by activation of metabotropic glutamate receptors (mGluRs) on the plasma membrane which are activated with synapses, forming a coincidence detector between calcium signals and synaptic activation (Nakamura, Barbara, Nakamura, & Ross, 1999; Nakamura, et al., 2000; Wang, Denk, & Häusser, 2000; Manita & Ross, 2010). Additionally, calcium waves are easier to initiate if there are higher cytosolic free calcium levels. For example, if calcium crossed the plasma membrane via VGCC activation during a bAP or via NMDA receptors activation from synaptic messaging. The role of RyR in calcium waves is much less clear and has only been observed in myocytes and not neurons (Luo, et al., 2008).

Calcium sparks are localized (about 2 µm), fast (about 45 ms), and stochastic events (Ouyang, et al., 2005; Berrout & Isokawa, 2009; Miyazaki, Manita, & Ross, 2012). The traditional form of the event is generated by RyRs. Their frequency is correlated with the subthreshold voltage across the plasma membrane and free calcium concentration in the cytosol. When a single RyR is activated, the nearby RyR are also activated due to CICR resulting in a strong local signal. But, the result is not a calcium wave and the signal does not propagate (Ross, 2012).

Calcium puffs are a form of calcium spark that is generated by IP3Rs. The main distinction between the two is that calcium puffs require raised cytosolic levels of IP3 to occur while sparks do not (Cheng & Lederer, 2008). Their frequency is modulated by the level of free calcium in the cytoplasm and not the subthreshold voltage as with standard calcium sparks. With a sufficient level of IP3 in the cytoplasm and high enough density of IP3R on neighboring ER segments then the calcium puff can activate regenerative and propagating CICR (Ross, 2012). Calcium sparks and puffs can interact with each other, causing changes in spread if both activate simultaneously (Miyazaki & Ross, 2013).

Calcium waves and calcium sparks are calcium events that originate from the ER calcium stores. IP3R channels initiated CICR is responsible for calcium waves which propagate across the dendritic tree. These events require high IP3 in the cytosol and will terminate in parts of the dendrites without sufficient IP3. On the other hand, calcium sparks are very fast and highly localized stochastic events that may be generated by either RyR or IP3R channels and involve only a minimal amount of CICR before stopping and thus do not typically spread. It is predicted that calcium waves and sparks, among other calcium events, combine with or complement the voltage- and ligand-gated transient currents along the plasma membrane, and are often referred to as the 'calcium toolkit' (Bootman, Lipp, & Berridge, 2001; Ross, 2012).

1.2.2 IP3 and Ryanodine Receptor Channel Distributions and Modulations

Calcium waves and calcium sparks are implemented by IP3R and RyR channels along the ER. The ability to generate either calcium event will depend greatly on the density and modulation of the channels. Their distribution is not uniform throughout the dendrites causing preferences for locations of the events. Additionally, through modulation their open probability is changed, effecting how likely they are to be open at a certain level of cytosolic calcium.

RyR and IP3R channels are both more abundant at branch points within the dendritic arbor, although they are present throughout the dendritic smooth ER (Walton, et al., 1991). This is witnessed in the fact that calcium waves preferentially initiate at branch points (Larkum, Watanabe, Nakamura, Lasser-Ross, & Ross, 2003; Fitzpatrick, et al., 2009) whether they are activated by synaptic input or metabotropic agonists (Nakamura, Lasser-Ross, Nakamura, & Ross, 2002). In fact, the region between branch points form "cold spots" where synaptically activated calcium wave propagation weakens (Fitzpatrick, et al., 2009) which further indicates a distribution where IP3Rs are denser at branch points. There must be significant quantities of IP3Rs along the ER between branch points as the calcium waves are able to travel along these regions even if they have trouble being initially generated there (Ross, 2012). Calcium sparks are much more predominant at branch points, indicating that RyRs are very tightly clustered at branch points with little or no density along the branches themselves (Ross, 2012).

RyRs have a large and complex modulation system that influence their open probability. While most of the studies utilize cardiac and skeletal muscle cells where RyRs play a critical role in the contraction of muscles (Ozawa, 2010), at least some of the regulation likely plays a role in neurons as well (Zalk, Lehnart, & Marks, 2007). RyRs have several sites which can be phosphorylated, including some by protein kinase A (PKA) and CaMKII, which increase the open probability, and thus the amount of calcium released in the presence of some calcium, as the amount of phosphorylation is increased. The channel is leaky when hyperphosphorylated even at resting levels of cytosolic calcium, and continuously releases small amounts of calcium from the ER into the cytosol. In addition to phosphorylation RyRs also associate with several proteins and enzymes. Calstablin1 (FKBP12), calstablin2 (FKBP12.6), and FK506-binding protein 1b/12.6 (FKBP1b) bind to RyRs and stabilize the close state of the channel and when not bound may cause RyR to become leaky. Calmodulin also binds directly to RyRs and acts to inhibit the opening of the channel, although this behavior is modified as calmodulin binds to calcium ions. Several other cytosolic calcium buffering proteins, such as calsequestrin (CSQ) and the histidine-rich calcium-binding protein

(HRC) also interact directly with the channels, likely to change its efficacy based on the cytosolic concentration of calcium (Zalk, Lehnart, & Marks, 2007; Ozawa, 2010; Gant, et al., 2011).

Both channels are located primarily at the branch points in the dendritic tree. However, IP3Rs are still found along the ER along the branches themselves. Correspondingly, calcium sparks are more likely at branch points and calcium waves preferentially initiate from branch points. Additionally, RyR channels have extensive modulation capabilities which change their open probability. Historic and recent calcium levels influence the probability of the RyR channels to open at different levels of the current calcium in the cytosol.

1.2.3 Specialized ER Structures

The ER contains specific channels so that it can carry calcium signals throughout the dendrites. Additionally, it also forms specialized structures so that its signals are amplified and/or interact with signals along the plasma membrane including: the spine apparatus (SA), lamellar bodies (LB), and subsurface cisternae (SSC). Examples of the last two are shown in Figure 1.1.

Within the head of some dendritic spines, a specialized form of ER is found called the spine apparatus (SA) (Gray, 1959). It consists of a series of stacked discs filled with calcium (Fifková, Markham, & Delay, 1983) that are connected to each other and to the remainder of the dendritic ER system (Spacek & Harris, 1997). It is thought that this structure plays a role in the long-term potentiation (LTP) and long-term depression (LTD) of the synapse (Jedlicka & Vlachos, 2008; Segal & Vlachos, The Spine Apparatus, Synaptopodin, and Dendritic Plasticity, 2010) along with calcium signaling within the spine head (Sabatini, Oertner, & Svoboda, 2002; Holbro, Grunditz, & Oertner, 2009). But the actual purpose is still under debate (Segal & Korkotian, 2014) although many studies have shown that it plays at least some role in learning since disrupting its formation causes mice to exhibit decreased spatial learning in the radial arm maze

(Jedlicka & Vlachos, 2008) and there is an increase in the number of spines with SAs after fear conditioning (Jasinska, et al., 2016).

The SA structure has a high density of RyR channels which serve to dump calcium from the internal SA/ER stores into the cytoplasm spine head when the synapse is activated, frequently with NMDA or AMPA receptors (Segal & Korkotian, 2014). This calcium release then plays a role in LTP/LTD of the synapse. The RyR channels in the spine head are colocalized with synaptopodin (SP) and it is likely that SP plays a strong role in forming the lamellar structure of the SA (Kremerskothen, Plaas, Kindler, Frotscher, & Barnekow, 2005; Segal & Vlachos, 2010).

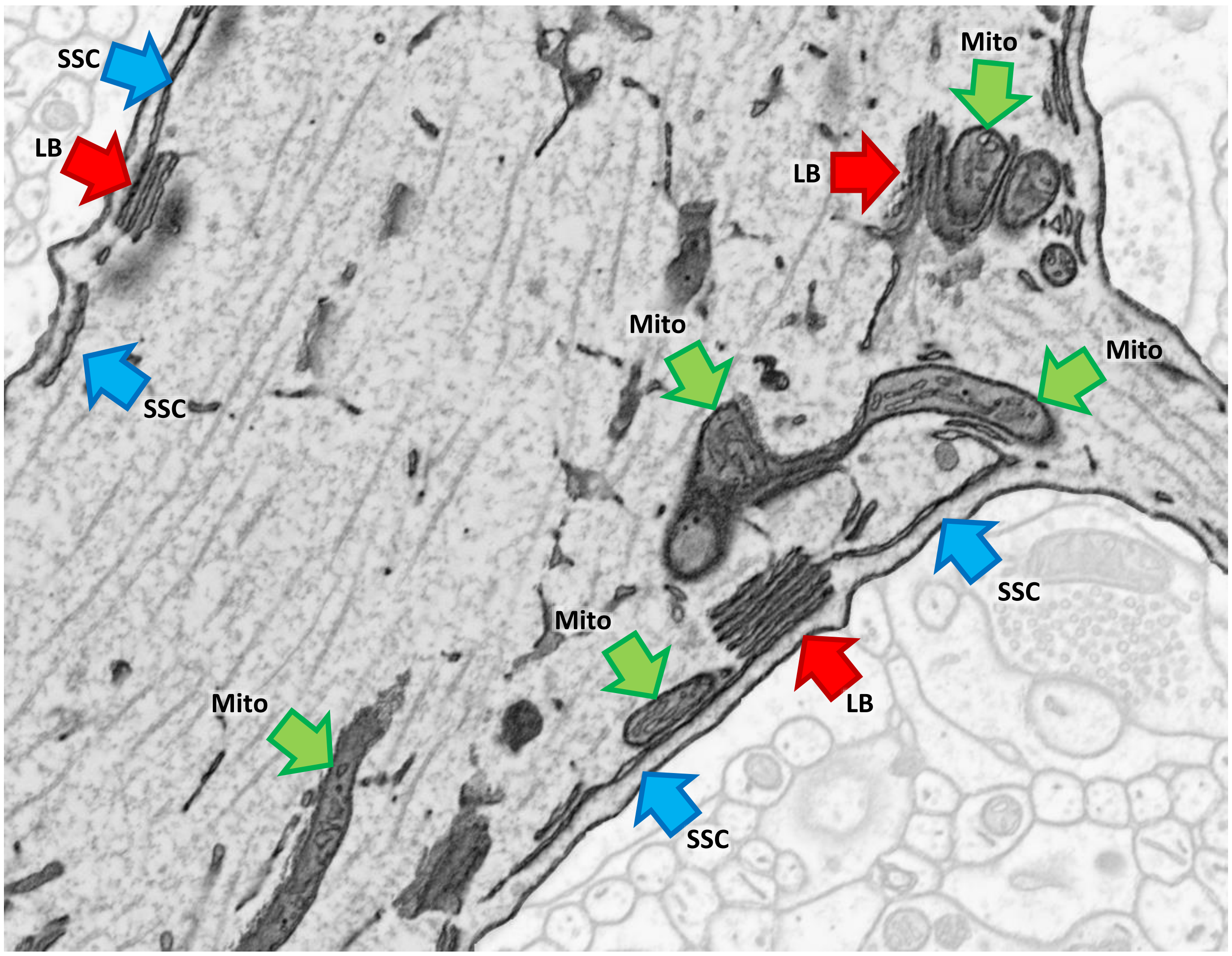

Figure 1.1. Lamellar bodies (LB) and subsurface cisternae (SSC) at a branch point of a Purkinje cell. Representative samples of each structure are indicated. Additionally, some mitochondria are labeled as well as “Mito”. The plasma membrane is the dark line surrounding the non-lightened region. The surrounding neuropil was manually segmented and lightened for clarity. Purkinje cell dendrite from the molecular layer of rat cerebellum acquired with SBFSEM on the Merlin 3View microscope at NCMIR, UCSD by Tom Deerinck.

Similar stacked disc formations, called lamellar bodies (LB), appear in other locations besides spine heads although they are significantly less studied (Le Beux, 1972; Broadwell & Cataldo, 1983; De Zeeuw, Hertzberg, & Mugnaini, 1995). LBs can be formed by local depletion of calcium (Garthwaite, Hajo, & Garthwaite, 1992) and activation of mGluRs (Banno & Kohno, 1998). These structures are also rich in RyR and IP3R channels. One of the predominant locations is at branch points in the dendritic arbor, as seen in Figure 1.1. In the figure, a related dendritic ER structure can be seen as well: the subsurface cisternae (SSC). The SSC is where the ER follows the plasma at a fixed distance. The name comes from the fact that they are a cistern of calcium (a.k.a. ER) that lies immediately beneath the surface of the cell (a.k.a. the plasma membrane) (Berridge, 1998).

1.3 Potassium Channels in the Dendrites that Influence Information Integration

While the calcium dynamics along the ER give rise to the inside cable of the cable-in-cable model, the outer cable, or plasma membrane, is also critical to information integration within the dendrites. Additionally, an important piece is intentional cross-talk between the two cables at specific locations under specific circumstances so that information can be passed between the two in a controlled manner. There are several plasma membrane based potassium channels that influence the integration of signals within the dendrites. Some of these are important for the passing of information between the two cable systems while others are important for the information integration performed by the plasma itself. Kv2.1 and Kv4.2 are two important potassium channels that open based on the voltage potential across the plasma membrane and are modulated based on historic and recent calcium activity. Both are modulated based on calcium activity and thus calcium events carried by the ER can influence their future behavior. Moreover, both channels appear to have special roles at branch points, where much of the integration of information in the dendrites is likely to occur, and Kv2.1 is tightly coupled with RyR channels on the ER. Additionally, the BK and SK potassium channels that are directly activated by calcium, and thus can be opened by signals carried by the ER. They also play roles in information integration within individual dendritic branches.

1.3.1 Calcium-Modulated Voltage-Gated Potassium Channel: Kv2.1

The Kv2.1 voltage-gated potassium channel (also known as KCNB1) is widely expressed throughout the brain (Misonou, Mohapatra, & Trimmer, 2005) and distributed over much of the dendritic arbor within individual neurons (Kirizs, Kerti-Szigeti, Lorincz, & Nusser, 2014; Bishop, et al., 2015). The following is a discussion of its role in the dendrites, its localization near LB and SSC, its interaction with RyR channels, and finally its modulation.

Kv2.1 is the primary delayed rectifier channel[3] in many neuron types which includes the pyramidal neurons in the cortex and hippocampus. Since the concentration of potassium ions is much higher within the cell (140-150 mM) than in the extracellular space (3-5 mM), the opening of potassium channels causes the positive potassium ions to flow out of the cell and create a net negative polarization effect on the voltage potential across the plasma membrane. Overall this means that activation of potassium channels causes an inhibitory effect on voltage signals being carried by the plasma membrane. Since they are voltage-gated, an increase of the voltage potential across the plasma membrane causes them to activate and subsequently dampen the effect, returning the cell to its negatively polarized state. This is one of the major differences between the excitability of axons and dendrites. They both contain similar amounts of voltage-gated sodium and calcium channels, however, there is a large difference in the number and type of potassium channels (Hoffman, Magee, Colbert, & Johnston, 1997). In particular, Kv2.1 channels have been shown to regulate the excitability of the dendrites during periods of high-frequency firing, ensuring

[3] A delayed rectifier channel does not activate until the membrane is positively polarized and then acts to reverse that by negatively polarizing the membrane. This behavior plays a major role in the shape of action potentials.

that the fidelity of the high-frequency signal is preserved and shapes the underlying cytosolic calcium density (Du, Haak, Phillips-Tansey, Russell, & McBain, 2000).

Kv2.1 channels are likely to be directly responsible for the formation of LB and SSC structures at branch points. The SP protein within spine heads is thought to be responsible for the formation of the lamellar SA structures but SP is not found outside of the spine necks in neurons (Wang, Dumoulin, Renner, Triller, & Specht, 2016). Elsewhere in the dendrites, the RyRs on the ER are tethered to Kv2.1 on the plasma membrane (Mandikian, et al., 2014). The stacked membrane discs of the LB contain clusters of RyRs on the ER which lie directly beneath large clusters of Kv2.1 channels on the plasma membrane (Rosenbluth, 1962; Du, Tao-Cheng, Zerfas, & McBain, 1998; Berridge, 1998; Mandikian, Cerda, Evans, Schneider, & Trimmer, 2012; Mandikian, et al., 2014). The tethering of the channels is done by the AMIGO-1 protein (amphoterin-induced gene and ORF [Open Reading Frame]) which is a plasma membrane that can associate with RyR or Kv2.1 independently. This means that if only RyR and AMIGO-1 proteins are expressed in a cell, RyR channels form SSC clusters (Mandikian, Cerda, Evans, Schneider, & Trimmer, 2012) and if only Kv2.1 and AMIGO-1 are expressed in a cell then Kv2.1 clusters form on the plasma membrane (Peltola, Kuja-Panula, Lauri, Taira, & Rauvala, 2011). When all three are expressed in a cell the colocalized clusters form spontaneously (Mandikian, et al., 2014; Fox, et al., 2015). This association is so important that when AMIGO-1 is artificially prevented from being created, or “knocked out”, the number of Kv2.1 channels synthesized is greatly reduced and a schizophrenia phenotype is displayed (Peltola, et al., 2016). Additionally, the Kv2.1 channels primarily show up in clusters within the dendrites and rarely as individual or small amounts by themselves. Furthermore, they are not found in spines or oblique dendrites (Kirizs, Kerti-Szigeti, Lorincz, & Nusser, 2014).

The Kv2.1 channel is found extensively phosphorylated (Misonou, et al., 2004; Murakoshi, Shi, Scannevin, & Trimmer, 1997; Shi, Kleinklaus, Marrion, & Trimmer, 1994). Dynamic changes in its phosphorylation state have been shown to fundamentally alter its overall expression amount, whether it is localized to the plasma membrane or sequestered, clustering within the plasma membrane, and its sensitivity to voltage across the plasma membrane of the channel (Cerda & Trimmer, 2010). The channel is rapidly dephosphorylated when there are large quantities of intracellular calcium; this has been seen *in vivo* in response to seizures (Misonou, et al., 2004) and hypoxia (Ito, Nuriya, & Yasui, 2010; Misonou, Mohapatra, & Trimmer, 2005) along with *in vitro* through various means that cause increased cytosolic calcium including IP3-mediated calcium release (Mohapatra & Trimmer, 2006). The primary effect of the mass dephosphorylation is the dispersal of the normal clustering of the channels which results in a much more uniform distribution of the channels across the plasma membrane (Misonou, et al., 2006; 2005; 2004). At least part of the dispersal of the channels is due to calcium release from ryanodine receptors reacting to zinc release which occurs during injuries (Schulien, et al., 2016).

The phosphorylation of Kv2.1 also affects its voltage dependence. It has several phosphorylation sites across the entire protein that are regulated by the protein calcineurin, whose activity is calcium-dependent. It has been observed that when more of these sites are phosphorylated a larger the voltage potential across the membrane is needed before the channel is opened, allowing for a graded response system (Mohapatra & Trimmer, 2006; Murakoshi, Shi, Scannevin, & Trimmer, 1997; Park, Mohapatra, Misonou, & Trimmer, 2006). Each phosphorylation site appears to have independent regulation, with some being regulated based on recent neuronal activity (Misonou, et al., 2006; Mohapatra, et al., 2009). The phosphorylation sites have complex interactions with each other (Tian, et al., 2004) suggesting that the Kv2.1 channel has an extremely intricate regulatory system. In addition, other phosphorylation sites on Kv2.1 have been identified that are not calcium-dependent including two sites that are regulated by AMP-activated protein kinase which acts to shut off energy-intensive processes when the amount of available energy in the form of ATP is low in the cell (Ikematsu, et al., 2011). The association of AMIGO-1 with Kv2.1 also acts to regulate the voltage sensitivity of Kv2.1 channels (Zhao, et al., 2014; Peltola, Kuja-Panula, Lauri, Taira, & Rauvala, 2011).

Ultimately Kv2.1 is a highly dynamic potassium channel with a large influence on its behavior coming from historic intracellular calcium levels. It is the largest contributor to keeping the voltage potential across the plasma membrane controlled. It is also tethered to ER-based CICR ryanodine channels. The Kv2.1 and RyR channels form clusters and colocalize with each other, particularly at dendritic branch points, creating LB and SSC structures within the ER. However, the Kv2.1 channel is primarily present near the soma and is rare in the distal dendritic branches (Kirizs, Kerti-Szigeti, Lorincz, & Nusser, 2014; Lim, Antonucci, Scannevin, & Trimmer, 2000). This means that while it may play an important role in calcium signaling, it only does so in the largest branches and only closest to the soma.

1.3.2 Calcium-Modulated Voltage-Gated Potassium Channel: Kv4.2

Another calcium-modulated voltage-gated potassium channel is Kv4.2 (also known as KCND2). It is expressed throughout the dendrites and is the dominant A-Type current[4] in dendrites (Duménieu, Oulé, Kreutz, & Lopez-Rojas, 2017). However, its distribution is not uniform throughout the dendrites in most neuron types. It forms clusters or puncta, it is only loosely associated with the location of synapses, and it is particularly abundant at the perimeter between the soma and the proximal dendrites (Alonso & Widmer, 1997; Jinno, Jeromin, & Kosaka, 2005). The channels can be re-distributed based on the spiking activity of the neuron thus changing the dynamics of the cell (Kim, Jung, Clemens, Petralia, & Hoffman, 2007). While this channel only has a minimal association with the ER and its calcium events, it critically regulates currents at branch points and is thus very important to dendritic information integration.

Kv4.2 has many roles in the dendrites, but they all involve various aspects of learning and interaction with calcium events. The primary role of Kv4.2 seems to be the dampening of excitatory voltage potential signals to novel stimuli, resulting in more consistent learning in various environments

[4] A-Type refers to the fact that the channels are voltage-gated, calcium-independent, potassium currents that have a characteristic rapid activation and inactivation.

(Kiselycznyk, Hoffman, & Holmes, 2012). This correlates to the fact that Kv4.2 is removed from the plasma membrane during LTP of synapses (Kim, Jung, Clemens, Petralia, & Hoffman, 2007; Chen, et al., 2006). The Kv4.2 channel also plays an important role at branch points throughout the dendritic tree by isolating voltage potential "plateaus" in different branches from each other (Cai, et al., 2004) and selective invasion of back-propagating action potentials (bAPs) into dendritic branches based on experience (Makara, Losonczy, Wen, & Magee, 2009; Migliore, Hoffman, Magee, & Johnston, 1999). These features at branch points show plasticity in that they can change over time with learning (Losonczy, Makara, & Magee, 2008). Kv4.2 also interacts with the calcium flux from IP3Rs on the ER, assisting to overcome the suppression they experience in thin dendrites due to the initial influx of calcium during calcium waves. This causes them to influence the latency and duration of calcium waves (Ashhad & Narayanan, 2013).

Like Kv2.1, Kv4.2 also has numerous phosphorylation sites and associated proteins which modulate its density on the plasma membrane and open probability. The phosphorylation by PKA and PKC (protein kinase C) decreases the activity of Kv4.2 and thus increases the amplitude of back-propagating action potentials (Schrader, et al., 2006). Phosphorylation by CaMKII, which is calcium dependent, causes the channel to become localized to the plasma membrane and thus prior intracellular calcium levels cause more Kv4.2 channels to be present in the plasma membrane (Varga, et al., 2004). Additionally, there are several voltage-dependent potassium channel-interacting proteins (KChIPs) that bind to the Kv4.2, including KChIP1, KChIP2, and KChIP3. These proteins are members of the NCS (neuronal calcium sensor) subfamily of calcium-binding proteins and thus are highly sensitive to cytosolic calcium levels (An, et al., 2000; Beck, Bowlby, An, Rhodes, & Covarrubias, 2002). They may even be responsible for migration of Kv4.2 in to and out of the plasma membrane (Duménieu, Oulé, Kreutz, & Lopez-Rojas, 2017). Additionally, NCS-1 itself is hypothesized to associate with and effect the open probability of the Kv4.2 channel (Guo, Malin, Johns, & Nerbonne, 2002). Finally, the protein neuritin causes upregulation, or increase in protein synthesis, of the channel (Yao, Zhao, Liu, & Chow, 2016).

Overall, Kv4.2 is a highly expressed and highly modulated potassium channel with critical roles in the plasticity of neurons. Its presence in the plasma membrane and its sensitivity to voltage are regulated in many ways, including several ways that respond to intracellular calcium levels. It is located in clusters and particularly concentrated at the perimeter between the soma and the proximal dendrites. It seems to have a pronounced effect at controlling voltage potentials between individual branches in the dendritic tree, and is thus likely more abundant at branch points, and with the propagation of back-propagating action potentials. While Kv4.2 does seem to have a special role at branch points in the dendritic tree and does manage the integration of information between branches, it has a minimal relationship with the ER. So, it is likely that its calcium-based modulation comes from other sources of calcium such as VGCCs on the plasma membrane which are activated during bAPs.

1.3.3 Calcium-Activated Potassium Channels: SK and BK

The final set of potassium channels to be discussed are not strictly voltage-gated like Kv2.1 and Kv4.2. They are instead mainly activated by direct association with cytosolic calcium. The two main channels are: 1) the small-conductance calcium-activated potassium channels (SK) which are completely insensitive to the voltage potential across the membrane and only open in the presence of cytosolic calcium, and 2) the large-conductance calcium-activated potassium channels (BK) which are independently activated by intracellular calcium and voltage potential (Vergara, Latorre, Marrion, & Adelman, 1998; Bond, Maylie, & Adelman, 1999)[5].

SK channels have many interactions with various calcium events and are associated with the SSC. They are specifically opened by IP3-mediated calcium waves in many neuron types (Ross, 2012) including

[5] Intermediate-conductance calcium-activated potassium channels (IKs) also exist, however there are no reports of them occurring in the dendrites of central nervous system but instead limited to axons (King, et al., 2015), enteric nervous system (Furness, et al., 2004), or outside neurons entirely.

CA1 pyramidal neurons (El-Hassar, Hagenston, D'Angelo, & Yeckel, 2011) and cortical pyramidal neurons (Yamada, Takechi, Kanchiku, Kita, & Kato, 2004; Gulledge & Stuart, 2005; Hagenston, Fitzpatrick, & Yeckel, 2008). They are responsible for hyperpolarizing the membrane after a calcium wave. However, if the calcium wave is isolated and does not propagate into the soma, then SK-mediated afterhyperpolarization is greatly dampened, indicating that they play a role in modulating communication going outwards from the soma. This additional level of information likely comes from R-type, L-type, and N-type VGCCs to which the SK channels are tightly coupled in the plasma membrane (and not the calcium channels on the ER) (Bloodgood & Sabatini, 2007; Rudolph & Thanawala, 2015; Marrion & Tavalin, 1998; Zhang & Huang, 2017). SK channels are also influenced by calcium sparks resulting in spontaneous transient outward currents (STOCs) across the plasma membrane. This implies that they are loosely coupled with RyRs on the ER. This effect is facilitated by, but does not require, the coupled VGCCs (Zhang & Huang, 2017). SK channels are also present in postsynaptic terminals and they are associated with calcium events within the spine heads, such as the events generated by NMDA receptors (Bloodgood & Sabatini, 2007; Jones, To, & Stuart, 2017; Ngo-Anh, et al., 2005; Zhang & Huang, 2017).

Recent simulations of calcium waves include a BK current to hyperpolarize the membrane after calcium influx (Neymotin, et al., 2015) but the effect is not actually seen experimentally in the central nervous system. In fact, there is little evidence that BK channels are activated by calcium waves in the dendrites of neurons in the central nervous system at all (Ross, 2012). They do play roles with calcium and ER within cultured sympathetic neurons (Akita & Kuba, 2000; Stocker, 2004), in parasympathetic cardiac neurons (Merriam, Scornik, & Parsons, 1999), and urinary bladder smooth muscle cells (Herrera, Heppner, & Nelson, 2000; 2002). However, the density of BK channels is uniform throughout the dendrites and may act to decouple the initiation zones for APs and bAPs during high-frequency inputs in cortical pyramidal neurons (Ramaswamy & Markram, 2015). Additionally, they are coupled to N-type VGCCs along the plasma membrane, less than 30 nm away (Marrion & Tavalin, 1998; Stocker, 2004).

Ultimately, SK channels likely play a role in various dendritic and postsynaptic calcium events while the role of BK channels is much less clear. These mainly associate with the plasma membrane-bound VGCCs and not the ER-bound CICR channels like RyR and IP3R. The behavior of both channels can be modulated but neither can be modulated through any pathway that involves calcium (Faber, 2009; Hoshi, Tian, Xu, Heinemann, & Hou, 2013). Additionally, they perform a distinct role from the Kv4.2 channels. The SK channels act to restore the resting potential within a single dendritic branch segment while the Kv4.2 channels act as “gate-keepers” at the branch points between different dendritic branches (Cai, et al., 2004). The Kv4.2 channels compartmentalize the dendritic arbor, allow each dendritic branch segment to act as a quasi-independent binary signaling unit using the SK channels and that signal is sent out to neighboring branches only when the Kv4.2 channels can be overcome. Thus, these collectively form the core system for synaptic integration within a single neuron, demonstrating the ability for a single neuron to act as a collection of several integrate-and-fire style neurons (Wei, et al., 2001; Cai, et al., 2004), and show that a single branch may act as its own computational unit (Losonczy, Makara, & Magee, 2008; Poirazi & Mel, 2001)

1.4 Dendritic Branch Point Morphology

So far, the discussion on dendritic information integration has been focused on the ion channels along the ER and plasma membrane, their interactions through calcium events, and the specialized ER structures related to them. However, the morphology of the dendrites, and in particular the branch points themselves, is an additional complexity which has a large impact on the transmission of signals through them and the overall input/output relationship of a neuron (Mainen & Sejnowski, 1996; Krichmar, Nasuto, Scorcioni, Washington, & Ascoli, 2002; Schaefer, Larkum, Sakmann, & Roth, 2003; Komendantov & Ascoli, 2009). The realization that branching patterns and sizes of the branches could affect the currents traveling through goes as far back as 1959 when Wilfrid Rall developed a cable-model of a dendritic tree that was measure experimentally and demonstrated that interrelations between unique morphology and specific electric properties of dendrites can be critical to the overall behavior of the neuron, challenging the dominant belief at the time that neurons were essentially uniformly implemented cells (1959; 1962).

When looking at a wave traveling along a conductor, such as a bAP traveling along the plasma membrane, the geometry of the branch point has a significant effect on the amplitude of the wave before and after reaching the branch. The geometric ratio (GR) of the branch will alter the maximal height of the wave at the branch point itself as well as within the branches into which the wave travels into. The geometric ratio is defined as:

$$GR = \frac{\sum_j d_j^{3/2}}{d_P^{3/2}}$$

where d_j is the diameter of the j^{th} daughter dendrite, and d_p is the diameter of the parent dendrite. In this case, the parent dendrite is not inherently the larger dendrite or the one closer to the soma, it is simply the dendrite from which the wave originates. Likewise, the daughter dendrites are the dendrites into which the wave will propagate into (Goldstein & Rall, 1974).

When $GR = 1$ then there is no significant difference in the height of the wave within the branch point or within the daughter dendrites as compared to the height while approaching[6]. This was the situation that was assumed to be true for each branch point within the dendritic tree prior to the work by Goldstein and Rall (1974). Some dendritic trees have a roughly consistent $GR = 1$, such as the motoneurons of cat spinal cord. However, most studied neurons do not have this property (Rall, et al., 1992).

However, when $GR < 1$, the overall diameter of the system is decreasing and there is a transient increase in the wave height within the branch point that persists for a short distance down the daughter

[6] If there are two daughter dendrites with equal diameter they must each have a diameter of ~ 0.63 times the parent for the result to be a $GR = 1$

dendrites. Eventually, its height returns to the height of the original wave. Additionally, there is a minor increase in the velocity of the wave within the branch point (Goldstein & Rall, 1974).

Conversely, when $GR > 1$ the overall diameter is increasing and there is a transient decrease in the wave height within the branch point. However, the wave may never return to the full height again within the daughter dendrites because if the decrease is too large then it may not be sufficient to sustain the regenerating wave. This means that the wave can fail to propagate into branches depending on the GR and the threshold required by the branch to propagate the regenerating wave. Even when it is a success, there is possibly a significant delay in transmission through the branch point (Goldstein & Rall, 1974).

The GR concept also applies to situations where there is a single daughter dendrite, essentially when the diameter of a dendrite significantly changes even when there is no branch point (Goldstein & Rall, 1974). These changes have been shown to be important in calcium dynamics of dendrites in simulations (Anwar, et al., 2014). In the extreme situation, when a traveling wave reaches a terminus, and thus $GR = 0$, there is a significant increase in the wave height due to the buildup of ions which are trapped on one side causing a reflection of the signal (Poznanski, 1999).

In a single detailed CA1 pyramidal cell model reconstructed from light microscopy data by D. A. Turner for Migliore *et al* (1999) the GR values range from 0.1 to 2.0. The max value of 2.0 arises from situations where a dendrite bifurcates into two equal-sized daughter dendrites. The small GR values are situations where there are large branches that decrease in size shortly after the branch from the trunk. The branch points with $GR < 1$ form two clusters: one near the soma (up to 50 µm away) and in the middle of the arbor (375-500 µm away). In the other regions (50-375 µm and more than 500 µm away) the GR values are all greater than 1. The most distal region (greater than 500 µm away) has the largest GR values, all of which are near 2.0. The large number of high GR at the distal ends prevent bAPs from getting to the furthest reaches of the tree in simulations (Acker & White, 2007).

The diameter of the dendrites varies greatly over the arbor and across different types of neurons. In general, going from thicker branches near the soma to thinner branches at the tips. The diameters range in size from about 4 µm in the proximal dendrites of Meynert visual cortex cells to 0.2 µm at the most distal tips on granule cells. In just CA1 pyramidal cells of rats they range from over 3 µm in proximal dendrites to 0.25 µm at the distal tips. (Stuart, Spruston, & Hausser, 2016). The diameter of spine necks, where the ER also follows, are even smaller, ranging in size from 0.04 to 0.51 µm with an average of about 0.2 µm but with no correlation to the distance from the soma (Arellano, Benavides-Piccione, DeFelipe, & Yuste, 2007; Harris & Stevens, Dendritic spines of CA 1 pyramidal cells in the rat hippocampus: serial electron microscopy with reference to their biophysical characteristics, 1989). This great difference in dendrite diameters has a large effect on the GR of the branch points and the disparity between the main trunk diameter and the smaller branches has been shown to be a key determinant in dendritic AP initiation and propagation (Migliore, Ferrante, & Ascoli, Signal Propagation in Oblique Dendrites of CA1 Pyramidal Cells, 2005).

Another major factor in the geometry of the branches is the surface area to volume ratio (SVR). For open cylinders with diameter d such as dendrites the SVR is $4/d$. Due to the inverse relationship, as the diameter decreases the SVR increases dramatically. The SVR could be $0.5\ \mu\mathrm{m}^{-1}$ near the soma it is much more likely to be around $16\ \mu\mathrm{m}^{-1}$ in the distal dendrites – a factor of 32 times higher. The reason that the SVR is so important is that ion influx into the cell is a function of the surface area since the ion channels are embedded within the plasma membrane while the ability for the ions to diffuse or be buffered by proteins in the cytoplasm depends on the volume (Anwar, et al., 2014). The SVR is then essentially the ratio of the ability for the cell to transport ions across the membrane over the ability for the cell to sequester ions away. Additionally, the effects of calcium events on the ER will be much more concentrated and have a larger impact on modulation or opening of potassium channels on the plasma membrane in regions with higher SVR, which is one of the contributing reason why calcium waves preferentially initiate at branch points leading into smaller dendrites (Ashhad & Narayanan, 2013). This increased effect of calcium events is also achieved by the LB and SSC which create constrained spaces for the calcium diffusion (Kuwajima, Spacek, & Harris, 2012).

All of this assumes that the dendrites are well approximated by a series of connected cylinders as per the multi-compartment and cable models. However, there is strong evidence that bifurcation is not a simple creation of two cylinders from a single cylinder. Instead there is a wide range of branch point shapes and the resulting branches do not always have “straight” sides but instead are frequently tapered. The taper influences the information passage into the daughter dendrites and the timing of information between the different branches. In particular, differences in the taper at a branch point can have as much effect as the modulation of Kv4.2 channels do on the propagation of signals outside of the branch and compartmentalization of signals within the branch (Ferrante, Migliore, & Ascoli, 2013).

The shape of a branch point is not necessarily consistent over time. During LTP the branch may gain a much wider taper, i.e. its base becomes significantly wider but remains the same overall diameter along its length (Park, Mohapatra, Misonou, & Trimmer, 2006; Ferrante, Migliore, & Ascoli, 2013). During glutamate reception it may become “lanky” and during spontaneous release become “husky” (Ferrante, Migliore, & Ascoli, 2013). This all represents forms of structural plasticity of the branch point (Bourne & Harris, 2008) changing the ability for the branch to convey signals to the rest of the neuron (Makara, Losonczy, Wen, & Magee, 2009). It is unsure if these structural changes are compensating, cooperating, or completely independent of other forms of plasticity such as increased Kv4.2 localization to the plasma membrane (Ferrante, Migliore, & Ascoli, 2013). Other structural changes occur in response to activity within the cells and overall learning of the animal including the diameter and length of spine necks, spine head diameters (Araya, Vogels, & Yuste, 2014), distribution of branch points within the arbor (Maravall, Koh, Lindquist, & Svoboda, 2004), and total length of the dendrites (Sorensen & Rubel, 2011).

Overall, the physical structure of the dendrites, and especially of the branch points themselves, is critical to the information processing performed by the dendrites. Even subtle changes in the taper of a branch coming out of a branch point can have dramatic effects on the integration of information within the dendrites. The morphology of the arbor is also not constant, and can go under rapid changes during learning, similar to the changes happening with the open probability and localization of the various ion channels on the plasma membrane and ER. Most of the studied changes have used light microscopy which does not necessarily have the resolution necessary to capture all the changes occurring and thus many details of the structural nature of the dendrites may still be missing.

1.4.1 Ionic Currents and Morphology at Branch Points Affect Propagation of bAPs

Several dendritic events, including calcium waves, are likely to be influenced by ion channel distribution and modulation along with dendritic, and possibly ER, morphology. However, difficulties in studying calcium waves and the ER directly have limited this research. The major dendritic event that has been significantly studied is the backpropagating action potential (bAP) and the complexities here are informative on what the cells can do with changes in morphology and ion channels. The ability for action potentials to actively propagate back into the dendritic tree, instead of passively diffuse, was first discovered by the Sakmann group within neocortical pyramidal cells (Stuart & Sakmann, Active propagation of somatic action potentials into neocortical pyramidal cell dendrites, 1994) and hippocampal CA1 pyramidal cells (Spruston, Schiller, Stuart, & Sakmann, 1995). Since the discovery of bAPs, they have been actively researched, particularly with respect to playing a role in learning, including STDP and LTP/LTD or synapses.

Not only do the bAPs propagate into the dendritic tree, but they have specific, activity-dependent, responses. This means that the ability for bAPs to reach the most distal dendrite branches is highly dependent on the recent number of APs generated by the cell. Additionally, when bAPs fail to reach some of the distal tips of the dendritic tree, the failure occurs at branch points in the dendritic tree. As bAPs travel further into the dendritic tree they attenuate, reducing in overall wave amplitude. When a burst of APs is generated within the soma of the neuron, frequently only the initial AP becomes a full bAP and the subsequent bAPs are greatly attenuated and more likely to fail at much earlier branch points. Only after a few seconds of no spiking activity can another AP invade most of the dendritic tree again. Ultimately, there is an activity-dependent system for allowing bAP invasion into the dendritic arbor and branch points act as the major gates (Spruston, Schiller, Stuart, & Sakmann, 1995). Additionally, bAP invasion of the dendritic tree is also dependent on recent activity of both inhibitory (Tsubokawa & Ross, 1996) and excitatory synapses (Stuart & Häusser, 2001; Magee & Johnston, 1997). Recent activation of inhibitory synapses decreases the likelihood of bAPs to reach nearby branches and recent activation of excitatory synapses increases the likelihood of a bAP to reach the nearby branches. This indicates that activity of both the neuron as well as its preceding neurons influence whether a bAP can successfully reach individual branches within the tree.

Besides the recent activity of various neurons, the morphology of the dendritic tree was found to play a role in the success or failure of bAPs at the distal tips. Minor variations in dendritic morphology alone can reproduce the range of bAP efficacies experimentally. The dendritic geometry of some neuron types is so effective that no physiologically-relevant modulation of channel densities could affect the propagation of bAPs while in other neuron types the channel densities combined with the morphology combine to form a more flexible system for regulating bAP invasion of the dendritic tree (Vetter, Roth, & Häusser, 2001). Within the same neuron types, different populations of neurons can develop that are either weakly-propagating or strongly-propagating. These populations have different AP shapes within the soma, indicating different modulation of sodium and potassium channels is occurring, but the populations also have a small but consistent difference in their branching complexity within a specific region of the dendrites (Golding, Kath, & Spruston, 2001).

Since bAPs are a large component of the STDP and LTP/LTD learning, the failure of the bAPs to propagate to some of the far distal branches may mean that synapses on the branches will always decrease in sensitivity. This happens because STDP relies on the arrival of the bAP within a certain time window of the synapse activation. If the bAP arrives within the window, the synapse strengthens. If not, then the synapse weakens. This means that if the bAP never arrives due to a failure to propagate to that branch, weakening (and thus LTD) will always occur. However, in some situations the reverse may occur. If several nearby excitatory synapses were active simultaneously it might be enough to cause the bAP to successfully reach a branch it may not have otherwise reached and possibly arrive earlier, thus changing something that may have been a depression of the synapse into a potentiation (Sjöström & Häusser, 2006). Additionally, bAPs have cross-talk with calcium waves and sparks, which means that a concurrent calcium wave may also increase the likelihood of a bAP reaching the far distal dendrites (Ashhad & Narayanan, 2013).

More advanced mathematical models of bAPs have been developed, including the traveling wave attractor model. This model simulates the bAP as a traveling wave instead of directly simulating the underlying ionic currents across the membrane. If the dendrites were uniform in channel properties and morphology, then the traveling wave would reach a steady state, which is called the traveling wave attractor. As this wave reaches regions which are not uniform the wave shifts from the steady state either in time or position, but eventually these shifts decay and the steady state will be acquired again. When using a model such as this the perturbations of various ion channel currents and morphology can be examined closely, and it can be easily seen which changes will cause failures of the wave to propagate. Using these models, it was determined that the Kv4.2 channel modulation is critical to the success of the bAP invasion, especially at branch points where they play a major role. The GR affects the success rate, even when the channel properties would otherwise support the active propagation. The interplay between these two effects is the ultimate determination if a bAP successfully reaches a particular dendritic branch (Acker & White, 2007). The relative roles of Kv4.2 and GR on bAP invasion of dendrites was confirmed with morphologically simple models. Like many previous studies, the maximal conductance of Kv4.2 channels was shown to be the largest determination in terms of successful propagation of bAPs, but the GR was influential when the Kv4.2 conductance was on the edge of success or failure. The GR significantly affected the timing of the waves, slowing down waves by up to 60% in the extreme cases (Bush & Kiggins, 2010).

The ability for a particular branch to change from weakly to strongly conducting waves occur with similar mechanisms to those underlying STDP. This form of plasticity is called branch strength plasticity (BSP), is carried out by the downregulation of Kv4.2 channels, and happens with NMDA-dependent manner like STDP does. However, its effect is not very strong unless there is also activation of muscarinic acetylcholine receptors (mAChRs) which are primarily activated during exploratory behavior of the animal, indicating that this form of learning is used during novel situations (Losonczy, Makara, & Magee, 2008). This is illustrated with rats that develop in a spatially-rich and socially-rich environment end up with a much higher proportion of strongly conducting branches than in control rats (Makara, Losonczy, Wen, & Magee, 2009).

The Kv4.2 channels that are regulated to control the ability for bAPs to invade far distal branches also play a role in the regulation of IP3R-mediated generation of calcium waves on the ER. This is especially prominent in the distal dendrites where the diameter is smaller, forcing the ER and the plasma membrane into greater proximity to each other. Overall there is a bell-shaped curve relationship between the amount of Kv4.2 and the ability for calcium waves to form. If there is too much Kv4.2 and the calcium waves are suppressed but if there is too little Kv4.2 then historic calcium levels build up to a high enough level to discourage further calcium events (Ashhad & Narayanan, 2013).

While other dendritic signals, such as calcium waves, are possibly more important that bAPs to intra-dendritic information passing, they are not as well studied as bAPs. Even though calcium waves have larger maximum calcium levels than bAPs, and thus are easier to detect with calcium indicating dyes, they have been studied for several reason. They were discovered in neurons more recently, spread throughout the dendrites slower, cause a much lower change in voltage potential across the plasma membrane, and require newly developed non-buffering low-affinity calcium indicator dyes to measure accurately (Ross, 2012). Also they are more difficult to model accurately using common simulation environments (Anwar, et al., 2014). There have been some recent successes in modeling them, but these have been made using simplified models (e.g. not modeling the ER directly) or estimations of parameters (Shemer, Brinne, Tegnér, & Grillner, 2008; Neymotin, et al., 2015).

Overall, branches and branch points have their own forms of plasticity and selectivity to the signals that can be passed into or out of them. The most studied examples of this involve bAPs, but at least some of the discoveries likely apply to other events as well. The ability of bAPs to invade the furthest distal dendrites is highly dependent on the amount of Kv4.2 current present, current excitatory and inhibitory synaptic activation, and to a lesser degree the morphology of the branches. Their plasticity mainly involves downregulation of Kv4.2 channels, which also influence calcium wave initiation, and is partially dependent on the animal being engaged in exploration of the environment.

2 Microscopy

There are several ways to get information about the structure of biological systems including MRIs, X-rays, crystallography, and atomic force microscopes. At the cellular level, light microscopy is a very common choice. However, it has its limitations and electron microscopy (EM) is needed to get high enough resolution of subcellular structures. In this section, the development and use of light and electron microscopy for imaging cells and subcellular structures is discussed.

2.1 Light Microscopy

Since the seminal work by Santiago Ramón y Cajal and Camillo Golgi (Golgi, 1906; Ramón y Cajal, 1906) light microscopy has become an indispensable tool for neuroscientists. In the early studies cells had to be darkly stained otherwise they were transparent. This was due to the limited resolution of the devices at the time which caused most structures to be unresolvable (Denk & Horstmann, 2004). However, if every cell in the sample was stained equally then the image would just be a mass of stain. The method Golgi discovered for staining neurons randomly stains only a small handful of neurons. The method for this selective staining is still unknown as of today, but is an ideal stain for studying the overall morphology of individual neurons.

These early stains only allowed for the visualization of the gross cellular morphology without any ability to study the structures within a single cell. As time progressed additional stains were developed that could stain specific subcellular structures, but frequently they stained too generally and subcellular components were still out of reach in light microscopy. The development of fluorescence microscopy and more sophisticated instruments eventually broke this barrier and allowed subcellular structures to be viewed with increasing levels of clarity and specificity (Wilt, et al., 2009). Many fluorescent dyes were discovered that can identify lipids (Greenspan, Mayer, & Fowler, 1985), specific ions (Grynkiewicz, Poenie,

& Tsien, 1985), DNA (Kapuscinski, 1995), and organelles (Buckman, et al., 2001). Additionally, a large breakthrough was made when fluorophores were attached to antibodies, as antibodies can be created that targeted most proteins (Coons, Creech, Norman, & Berliner, 1942). This allowed for specific proteins and subcellular structures to be labeled. Eventually the introduction of genetically encodable fluorescent proteins, such as green fluorescent protein (GFP), allowed proteins to be precisely targeted with a covalently linked fluorescent tag without the need for additional staining (Tsien, 1998; Giepmans, Adams, Ellisman, & Tsien, 2006). While the labeling techniques were improved, the instruments used to acquire the images were also improved. Differential interference contrast (DIC) microscopy was developed allowing viewing of unstained samples by integrating two light rays that are slightly offset and have different polarizations from each other (US Patent No. 2924142, 1952; Lang, 1968). Confocal scanning laser microscopy allowed thicker samples to be used and for only a single plane of that sample to be imaged, allowing for the construction of 3D images (Davidovits & Egger, 1969; White & Amos, Confocal microscopy comes of age, 1987). Subsequently two-photon excitation microscopy, which uses a method much more complex than confocal microscopy, was developed and improved on its results by allowing for deeper tissue penetration and reduced phototoxicity (Denk, Strickler, & Webb, 1990). All of these improved the clarity of results and depth-of-field but were still limited in field-of-view. Using mosaicking a significantly wider field-of-view can be provided by serial acquisition (Chow, et al., 2006).

Light microscopy methods have several benefits such as the ability to use multiple labels in a single sample, the viewing of living samples that can be reused, and the creation of results of sufficient quality with relatively non-specialized equipment. However, only structures that have been specifically labeled or have unique light diffraction properties can be viewed and the appropriate antibodies or genetic strains are required to mark the structures. Additionally, the diffraction limit of light is around 250 nm depending on the wavelength and numerical aperture which means that light microscopy cannot resolve objects smaller or closer together than that (Abbe, 1873). Moreover, 250 nm is much larger than several subcellular components such as ribosomes (25-30 nm in diameter), synaptic vesicles (about 40 nm in diameter), and ER (50-100 nm in diameter). It is even larger than some cellular protrusions such as spine necks and the finest dendrites (around 200 nm in diameter). There are new super-resolution light microscopy techniques that have broken past the diffraction limit and can theoretically obtain resolutions in the tens of nanometers (Gustafsson, 2000; Betzig, et al., 2006; Rust, Bates, & Zhuang, 2006). These methods have been used to get detailed localization of synaptic cytoskeletal filaments (Pielage, et al., 2008) and receptor proteins (Dani, Huang, Bergan, Dulac, & Zhuang, 2011). They are however still in development, only work with fluorescently labeled samples or subcellular structures with specific optical properties, and frequently have slow acquisition rates.

2.2 Electron Microscopy

Electron microscopy is still the most common choice because it can obtain even higher resolutions, has established preparation techniques, and does not require specific labeling using a fluorophore. However, electron microscopy does have several drawbacks including the destruction of the sample being imaged, only acquiring a single channel of information, the complete dehydration of the sample, and the necessity of staining.

In electron microscopy, a stream of electrons is “fired” at the sample, just as in light microscopy a stream of photons is “fired” at the sample. Using electromagnetic lenses, the electrons are properly focused, similar to the glass lenses and other optics in a light microscope. The electrons being fired at the sample are more likely to interact with atoms that have large nuclei, such as heavy metals. Since these are rare in organic systems (significantly less than 0.1% of the total atoms in the body), with the largest common element being calcium with a mass of 40 amu, the samples are stained with heavy metals such as osmium (190 amu), lead (207 amu), and uranium (238 amu). Different chemical substances and cellular structures have different affinities for the various metals, and therefore the selection of the heavy metals used will influence which structures will be stained and visible. Additionally, specific labeling of proteins can be done using heavy metals attached to antibodies, with the most popular being immuno-gold labeling, or using genetically encoded singlet oxygen generators.

There are two main different types of electron microscopy: transmission electron microscopy (TEM) and scanning electron microscopy (SEM).

In TEM the electrons are fired through the sample and the electrons which are not deflected by or otherwise interact with the sample (called the transmitted electrons) are recorded by using a fluorescent screen, photographic film, or specialized CCD camera. The entire sample is recorded at once. Larger nuclei deflect more electrons and thus let fewer electrons through to be imaged. The process results in acquiring the sum of densities along a particular path through the sample, with the brightest spots being the location where there are no large nuclei between the electron source and the detector and the darkest spots being the greatest number of large nuclei between the electron source and the detector. Thicker samples require electrons with higher energies and velocities to be able to penetrate the sample and reach the detector. However, higher energy electron beams cause beam-induced mass loss, resulting in a destruction of the sample. Additionally, the thicker the sample, the more loss there is due to electron scattering and chromatic aberration. Also, since the detector 'sees a sum' across the entire sample, thicker samples likely result in lower signal variation over the entire sample. Typical sections are 50-100 nm thick which is thinner than many organelles, but this is useful for imaging small organelles such as mitochondria (Palade, The fine structure of mitochondria, 1952), ribosomes (Palay & Palade, 1955), and endoplasmic reticulum (Palade & Porter, 1954).

In SEM, a much lower intensity beam is used that only penetrates the top layer of the sample, typically only a few nanometers deep. The interaction of the electron beam with the atoms in the sample produces secondary electrons which are detected. These are electrons generated from the ionization of the atoms along the surface of the sample; they are not the same electrons shot at the sample, which would be primary electrons. Additionally, some SEMs are designed to detect backscattered electrons (BSEs): primary electrons that are reflected back towards the beam source. The beam is highly focused onto a single point on the sample, typically only a few nanometers in diameter. The beam is moved in a raster scan to cover the entire sample, recording the number of electrons generated or reflected at each spot as it moves. The result is an entire 2D image. As a result, the SEM can have an extremely wide field of view but larger fields of view take more time. Typically, the spatial resolution is about ten times less than TEM.

Both of these base techniques generate 2D images only, either a projection of the whole thickness of the sample or from just the top few nanometers. However, both of these techniques have been utilized in various ways to acquire 3D data volumes.

The most straight-forward way to extend these is by using them multiple times on consecutive slices of the sample. This was first done with TEM and called serial section TEM (ssTEM) with multiple slices being taken from the sample and placed on a grid, maintaining their order (Gay & Anderson, 1954; Sjöstrand, 1958). This technique was used to complete the very first connectome, that of *C. elegans* (White, Southgate, Thomson, & Brenner, 1986). Even though this method is still used today, it is extremely labor intensive, requiring significant time by highly trained experts during each step including preparation, section cutting and collecting, imaging, and reconstruction. Since each step requires human interaction, there is a large risk of human error causing much of the data to be unusable.

The development of serial block-face SEM (SBFSEM) alleviated some of the problems with ssTEM. When moving the concept from TEM to SEM, the slicing procedure was made automatic and built into the microscope imaging chamber itself (Leighton, 1981). This is possible since the entire sample block, without being cut into slices, could be placed within the SEM where the top is imaged and then sliced off the top of the block. The imaging and slicing is then repeated, generating a series of 2D images that can be stacked into a 3D volume. With this technique, expert involvement was no longer required for section cutting and collection and greatly reduced during imaging. However, a problem was discovered: since the sample block was being constantly bombarded with electrons with some of them becoming trapped within the top layer, the block would build up a significant negative charge. This negative charge would slow down other electrons being shot at the surface, quickly reducing the ability to accurately image the surface of the block. Due to this problem and because computer hardware was not capable of storing the large amount of data that was being generated, the idea was mostly abandoned at the time.

The concept was revitalized in 2004 by utilizing an environmental SEM (ESEM). Unlike traditional SEMs, ESEM does not maintain a complete vacuum in the sample chamber. The low concentration of gas produces positively charged ions when the electron beam interacts with it and these positively charged ions mitigate the negative charge buildup on the surface of the sample block (Denk & Horstmann, 2004). This version of SBFSEM also uses BSE detectors, creating excellent contrast when combined with traditional TEM stains, allowing a smoother transition from ssTEM to SBFSEM. The downside of this method is that the lack of a complete vacuum in the sample chamber causes scattering of electrons within the gas chamber, both of the electron beam and those reflected from the sample, causing a significant reduction in signal-to-noise ratio. To overcome this, the electron beam can dwell longer on each point on the surface of the sample. This means data acquisition is slower and may produce additional damage to the sample. Other solutions for reducing the surface charge problem while maintaining a high vacuum have been developed since then, including automated coating of the sample block after each cut with a thin conductive metal layer (Titze & Denk, 2013) and special staining techniques (Deerinck, et al., 2010).

One remaining issue with SBFSEM is that the axial resolution (the distance between individual slices of the sample block) is 25-100 nm/slice which is almost always an order of magnitude higher than the lateral resolution (the recording size of the electron beam within the plane of the surface of the sample block) (Peddie & Collinson, 2014). Many automatic segmentation techniques require the data to be isotropic meaning the spacing in each direction is equal (Sommer, Straehle, Kothe, & Hamprecht, 2011).

Additionally, some structures, such as ER, are difficult to follow with slices thicker than 20 nm. To overcome this problem, focused-ion beam SEM (FIBSEM) was developed. In this method, instead of cutting the top layer off the sample block as in SBFSEM, the top layer is removed by ablation with a focused beam of gallium ions. Using this method, an axial resolution of 3-15 nm is attainable (Peddie & Collinson, 2014), providing isotropic datasets. The drawback of this method is that acquisition time is greatly increased and the field of view is substantially limited compared to SBFSEM (Wei, et al., 2012; Knott & Genoud, 2013; Peddie & Collinson, 2014). Thus, for this project, SBFSEM was utilized to ensure that a large field of view could be acquired that included several dendritic trees instead of at most a single tree.

Electron tomography (ET) is another way to expand the capabilities of electron microscopy. In this case it is based on TEM. It does improve the resolution of the acquired data, including allowing for isotropic image voles, using a completely different technique from the serial methods described above. But in doing so it sacrifices the possible field of view afforded by SBFSEM. To increase its field of view, serial sectioning concepts can be applied to it, thus serial-section ET (ssET) can be performed as well. However, this becomes extremely labor intensive and infeasible to perform on regions larger than a single cell body. Thus, while this would increase the resolution of the data and give isotropic volumes, to achieve a wide field of view the SBFSEM technique is used for this project.

3 Segmentation

The acquisition of images is only the first step in getting quantitative information from a sample. The images must then be segmented. Segmentation is the process of breaking up images into the sets of pixels that represent each of the distinct objects within the image. For example, when looking at an image of a dense region of neuronal dendrites, unmyelinated axons, and glial cells, called neuropil, we may want to segment each of the neurons in the image and the various subcellular organelles. Once the pixels are broken into these groups, metrics for each group can be calculated, such as the percent of the neuron that is filled with mitochondria, the shape of nuclei, or the position of ER relative to the branches within the dendritic tree.

This chapter begins with a look at the historic evolution of segmentation which progressed from creating physical models, to manual tracing on a computer, to the modern-day usage of crowd-sourcing to create large-scale models. Then it moves on to the general process of performing automatic segmentation on image stacks. Finally, it examines the details of the assisted segmentation algorithm Livewire and the automatic segmentation algorithms CHM-LDNN and GLIA used in this project. A custom Livewire implementation was created for this project while CHM-LDNN was completely rewritten to improve its accuracy and efficiency. Also included in this section are the various tools created for the project to create a pipeline so that data can more readily be taken from microscope to practical and quantitative models.

3.1 Introduction

The technological advances in electron microscopy have led to more automated systems with increases in resolution, resulting in very large neuronal tissue image volumes at the level of ultrastructural, and sometimes molecular, resolution with decreasing amounts of work by skilled professionals. This has resulted in the exponential growth in the acquisition of image data (Peddie & Collinson, 2014). While this has provided the community with vast quantities of raw data this does not necessarily equate to readily usable data. As Stephen Senft put it "the bottleneck to reconstruction from each epoch has been processing speed, data access and, most importantly, processing intelligence" (Senft, 2011). While technological advances in computer hardware have increased at a near-exponential rate, so that processing speed and data access have kept up with the acquisition rate of images, the processing intelligence, which he mentions as the most important, has not increased at the necessary rate.

Typically, one prefers quantitative data which gives measurable values, numbers that can be gathered in a predictable manner and whose statistical significance can be shown, as opposed to qualitative data. Typically, the goal of microscopy is to gain an understanding of the objects within the view, the size and shape of each of these objects, and their relative positioning to other objects. A straightforward method to accomplish this is stereology (Gundersen, et al., 1988; Vanhecke, Studer, & Ochs, 2007; Rigoglio, et al., 2012). For example, when a set of points is chosen from the dataset and each point is classified as a particular object. As long as there is not a systematic bias in choosing the points and with a large enough sample of points the results can give fairly accurate results for total area or volume that each object occupies and potentially some spatial relationship between objects. Other forms of stereology involve measuring the area of objects on several individual planes and then computing a volume from the combinations of the areas, or similarly measuring the outline of objects on planes and getting the overall surface area of the object.

A bit more advanced method is image segmentation. The entire image is segmented (i.e. divided into sets of pixels) and every point is classified as part of an object or not. Many of the stereology techniques apply to the analysis of the objects after they are segmented, however in stereology only a subset of the points are processed while the goal in segmentation is to classify every point (Zhang Y.-J. , 2006). Segmentation is thus even more labor intensive than stereology, which is labor intensive in itself to get usable results.

Historically, the raw image data was manually processed with extremely labor-intensive methods. Until the arrival of modern computers physical models would be constructed from the raw images that could then be studied by physically manipulating them. Several methods of converting serial micrographs into 3D models were used. One of the earliest methods was tracing structures onto transparent cellophane sheets and gluing them together (Bang & Bang, 1957; Sjöstrand, Ultrastructure of retinal rod synapses of the guinea pig eye as revealed by three-dimensional reconstructions from serial sections, 1958). Later researchers projected the images, traced the structures, cut them out, and glued them together into stacks using polystyrene (Pedlar & Tilly, 1966), white cardboard (Roberto, García, & Wettstein, 1973), or graph paper (Hoffmann & Avers, 1973). Some groups even created models out of string wrapped around the perimeter of the structures (Braverman & Ken-Yen, 1983). While these models were frequently easier to deal with than a series of 2D images, they were also frequently large and required additions to prevent collapsing. Additionally, it was difficult to gather quantitative data from these models and the time required to create them limited the number of structures that could be analyzed.

As computers became more capable, more ambitious imaging projects began using the emerging technologies to create 3D models of structures within EM data (Levinthal & Ware, 1972; Macagno, Levinthal, & Sobel, 1979). Since early computer hardware was quite limited the system could not handle displaying the raw image data itself. Instead the data had to be traced using special digitizing pads. These systems provided the ability to calculate basic morphological parameters (Cowan & Wann, 1973; Fox, Rafols, & Cowan, 1975; Moens & Moens, 1981; Prothero & Prothero, 1982) and 3D rendering (Willey, Schultz, & Gott, 1973; Stevens & White, 1979; Macagno, Levinthal, & Sobel, 1979; Moens & Moens, 1981; Braverman & Braverman, 1986). There were several complications with these systems however, including the fact that some systems required multiple tracings, once to trace the raw images and another to trace using the digitizing pad (Braverman & Braverman, 1986) and that in none of the systems was the final digitized output ever shown on top of the image data and thus mistakes could easily be missed. Moreover, errors could be introduced by not properly aligning images. This was addressed by some groups by either allowing a researcher to view multiple images at the same time with special optics (Levinthal & Ware, 1972) or played as a filmstrip (Harris & Stevens, 1988). While these methods were used to complete reconstructions for the first connectome, that of the nematode *C. elegans*, the process took several years and the nervous system contains only 302 neurons which are essentially invariant between individuals (White, Southgate, Thomson, & Brenner, 1986). These methods would clearly not scale to the 70 million neurons in the mouse brain (Herculano-Houzel, Mota, & Lent, 2006) or the 86 billion in the human brain (Azevedo, et al., 2009), both of which vary greatly between individuals.

As computer hardware progressively improved over time, the software for segmentation of EM data also improved, adding new features for reconstruction of objects and their visualization (Young, Royer, Groves, & Kinnamon, 1987; Allen & Levinthal, 1990). Two of the early available packages were CARTOS II (Allen & Levinthal, 1990) and the SYNU suite (Hessler, et al., 1992). These allowed the creation and display of multiple 3D meshes from segmentations and enabled creation of movies allowing for improved interpretation of the 3D models by viewing from multiple angles. Shortly afterwards, in 1994, the IMOD package was released which contains several programs for working with reconstruction of 3D models from many three-dimensional imaging sources including a wide variety of EM sources such as TEM and SSSEM (Kremer, Mastronarde, & McIntosh, 1996). Due to its high versatility, numerous powerful features, and continual updates it has remained in frequent use to this day (Medeiros, et al., 2018; Pichat, Iglesias, Yousry, Ourselin, & Modat, 2018; Kim, et al., 2018). IMOD allows users to "trace" or draw contours around regions of interest using various input devices, segmenting the regions. These contours can be "meshed" or joined across several images in a stack of related images to form 3D meshes of the segmented objects. Each of these objects can be displayed individually or in a view with other objects, allowing them to be visually analyzed individually or in relationship to other objects. Additionally, it supports quantification of some morphological properties of the 3D objects.

Even though IMOD has fairly widespread use, several other software packages have been developed to fill various niches or improve specific features for specific purposes or workflows, including Xvoxtrace (Perkins, et al., 1997), UCSF Chimera (Pettersen, et al., 2004), XMIPP (Sorzano, et al., 2004), Reconstruct (Fiala, 2005), Bsoft (Heymann & Belnap, 2007), EMAN2 (Tang, et al., 2007), Viking (Anderson, et al., 2010), KNOSSOS (Briggman, Helmstaedter, & Denk, 2011), TrakEM2 plugin for ImageJ (Cardona A. , et al., 2012), NeuroMorph plugin for Blender (Jorstad, et al., 2015), and Microscopy Image Browser using MATLAB (Belevich, Joensuu, Kumar, Vihinen, & Jokitalo, 2016).

All of these options, including IMOD, focus on manual or assisted segmentation. Manual segmentation is when a human traces around each region of interest. As time progresses and computational power is more readily available, the computer programs themselves have begun providing real-time feedback to assist in the segmentation process being performed by the human, reducing the amount of human time required to be able to process the same amount of data. Two examples of assisted segmentation techniques are Livewire (Mortensen, Morse, & Barrett, 1992) and Interactive Merging (Jones, Liu, Ellisman, & Tasdizen, 2013; Liu, Jones, Seyedhosseini, & Tasdizen, 2014). However, even with modern programs, the amount of microscopy data being generated has far outpaced the relatively minor increase in speed at which humans can process the data, even with the help of computers. This has resulted in several possibly important datasets just being abandoned or only a small subset of datasets being used in studies.

During this same time several automated techniques for analyzing images have been developed as well. However, they have not been widely adopted for a variety of reasons. First, until recently, they have suffered from not being nearly as accurate as humans are. This means that all automatic results need to be fully proofread by humans and corrected for mistakes, reducing the overall amount of trained professional time but not eliminating it or even making it constant relative to the amount of data to process. Neuropil EM data is especially difficult for automatic segmentation because of the complex ultrastructural cellular textures and the considerable texture variations in shape and physical topologies within and across image sections (Jain, et al., 2007; Lucchi, Smith, Achanta, Lepetit, & Fua, 2010). The second major issue is that the automatic segmentation solutions developed are not readily accessible or usable by the standard lab or the non-computer scientist. They usually require specialized hardware with large amounts of memory and are not straightforward to setup with little in the way of documentation for the common scientist.

Due to these issues, most neuroscientists utilize the manual/assisted segmentation route when processing their data. This works for small studies but prevents any sort of large-scale reconstructions without significant investments in manpower and time. Even when using common short-cut techniques such as interpolation of contours between images and making assumptions that organelles are spheres or cylinders, manual/assisted segmentation remains incredibly laborious (Noske, Costin, Morgan, & Marsh, 2008). Even segmentation of the cell membranes in small regions of a the visual system of the *Drosophila melanogaster* fruit fly were reported to take several months to years in terms of continuous labor (Chklovskii, Vitaladevuni, & Scheffer, 2010; Plaza, Scheffer, & Saunders, 2012). Briggman and Denk estimate that a single mouse cortical column would take 10000 people working for one year to trace all of the neural connections (2006). It is estimated that manual/assisted reconstruction of all of the mitochondria within a single mouse brain in one year would take every citizen in the city of Chicago working nonstop in parallel while the task for a human brain would require every person on the Asia and Africa continents (Perez, 2014; Perez, et al., 2014).

3.1.1 Distributed Manual Segmentation

While these estimates would imply that any effort to perform such a reconstruction without most of the work being automated would never succeed, there have been attempts to distribute the manual segmentation work among people for smaller regions instead of whole brains. One of the first major efforts was by Kevin Briggman and Winfried Denk who employed over 200 trained individuals to trace just the skeleton of neural processes to establish a wiring diagram of the entire retina (Briggman, Helmstaedter, & Denk, 2011), a much simpler task then segmenting organelles or the cellular membranes. Large groups of undergraduate students have also been used to perform segmentations of subcellular organelles (Perez, et al., 2014).

Both of these approaches do not scale well as all of the individuals must be trained before their work can be trusted without professional review (Perez, et al., 2014). Additionally, these approaches typically have the individuals use the same software that the professional tracers utilize and thus there are potential setup costs and difficult learning curves since these software packages are typically designed for professionals.

Other groups take more scalable approaches by either employing micro-laborers or by gamifying the process, collectively called crowd-sourcing. To use either of these, the training process for the individuals must be greatly reduced as any lay-person will be encouraged to help with the process. To accomplish this the tasks asked for the individuals to complete must be changed, making them simpler and faster, along with the interface being simplified, with a goal of being usable without any sort of installation to reduce the startup cost to nothing. Also, instead of having a professional review all of the results, the workers are monitored in an automated fashion, by having them complete tasks already completed by other individuals, perhaps even professionally completed tasks periodically.

The most famous of these approaches for neuroscience is with Sebastian Seugn’s EyeWire launched in December 2012 (Kim, et al., 2014). This simplifies the task to deciding if two neighboring regions should actually be merged into a single region or not. The initial regions are generated using automatic segmentation that is designed to greatly over-segment the data, i.e. it breaks everything into pieces that are too small, attempting to guarantee that no region contains multiple neurons by allowing each neuron to be in multiple regions. The entire process is run as a game with rankings, levels, and virtual rewards (Tinati, Luczak-Roesch, Simperl, & Hall, 2017). In the first 2 years they claimed that 100 neurons had been completed and after 5 years 3,000 neurons had been completed (excluding their axons) (Sterling, 2017). This amount of data resulted in several scientific discoveries about the wiring in the retina (Kim, et al., 2014; Greene, Kim, & Seung, 2016). Moritz Helmstaedter's Brainflight is a similar system that uses a mobile phone app where users also connect neighboring oversegmented regions in a highly graphical game with reward mechanisms to keep users playing (Marx, 2013).

Besides making a game out of the process, some groups have taken to using the micro-labor market. The DP2 platform uses Cytoseg to perform an oversegmenation and then has users, through Amazon Turk, decide if two points lie within the same cell or not. Instead of fancy graphics and virtual rewards, the workers are instead paid for each task completed (Giuly, Kim, & Ellisman, 2013). Another group asked users to pick out the correct spherical objects from an automatic segmentation (Lee, 2013).

Overall, while there have been advances in using computers to assist and accelerate the process of segmentation done still primarily hand-drawn. Even in the most extreme cases, humans are still critical to the process, but systems are being developed to reduce the amount of training an individual needs to help segment the data so that lay people can contribute to the process. However, this is ultimately not scalable to entire brains of even mice, and especially not for humans, as it would require so much manpower and better, more automated, solutions are required.

3.1.2 Automatic Segmentation

Better algorithms need to be implemented in order to do truly automatic segmentation. Current solutions are beginning to reach the accuracy level of humans, i.e. if multiple humans segment the same data, the error between them is about the same amount of error generated by the cutting-edge automatic segmentation algorithms. Most of these algorithms utilize supervised machine learning. Supervised machine learning means that the algorithm “learns” based on a small training dataset that has been manually segmented by a human and then applies that knowledge to the rest of the dataset. The algorithm goes through two phases: **training** which is when the algorithm is learning what to segment and **testing** which is when the algorithm is applying what it has learned to novel data.

Machine learning systems utilize sets of feature vectors and labels. During the training phase, both a set of feature vectors and labels are provided to the algorithm and it learns to associate each of the feature vectors with the corresponding label. During the testing phase, only feature vectors are provided to the algorithm which then produces the predicted set of labels based on what it had learned. This phase is also called classification as it aims to classify each of these new feature vectors based on its previous experience. When performing image segmentation, each feature vector represents one pixel in the image and is composed of many values that represent the basic “features” of that pixel. A feature is a numerical value that represents some statistic or information about the pixel; for example, how edge-like the pixel is or the size of the largest similar-intensity region containing the pixel. The features must be chosen carefully so that they are informative about whether a pixel is part of a particular type of object. The labels are then the type of object being segmented or possibly a binary answer of whether the pixel is part of the learned object or not.

One limitation of such an algorithm is that it will only be able to recognize what it has been taught and will not be able to easily extrapolate this knowledge to other situations. This means that fluctuations in the data caused by variations in the staining procedure or image acquisition properties cause enough differences so that the algorithm can no longer accurately apply what it learned to the new dataset. Every organelle and every dataset will need its own manually segmented training dataset and run through the training process before it can be used to test the remainder of the dataset. This means that it cannot be used on images that are coming directly from a microscope since the manual segmentation of the training dataset is required.

However, it is feasible that as more and more training sets are created a specific training set will not need to be manually created for every new image set. Instead the algorithm would re-use the training data, possibly combined from multiple previous datasets, to form a new training set automatically. This transfer of knowledge would begin to make the algorithm an unsupervised machine learning algorithm, which works without any manual direction from a human supervisor.

The primary drawback to the modern algorithms, besides the need to manually create training datasets, is the computational resources that they require in terms of both memory and time. Most currently require so much memory that they cannot be run on high-end computer hardware available to general labs but instead require supercomputers. The amount of time required would make it impossible to perform in real-time at the speed microscopes are able to generate data. Finally, most of these algorithms are developed in academic settings and are not created for use by the average researcher in the average lab.

There have been concerted efforts to improve the accuracy and usability of automatic segmentation techniques for neuroscience. This includes the IEEE International Symposium on Biomedical Imaging (ISBI) 2012 and 2013 competitions to automatically segment 2D and 3D EM neuropil data (Arganda-Carreras, Seung, Cardona, & Schindelin, 2012; Arganda-Carreras, Seung, Vishwanathan, & Berger, 2013; Arganda-Carreras, et al., 2015) and the SLASH (Scalable system for Large data Analysis and Segmentation utilizing a Hybrid approach) group. The SLASH group is a collaboration between the National Center for Microscopy and Imaging Research (NCMIR) at UCSD and the Scientific Computing and Imaging

Institute (SCI) at the University of Utah to develop automatic segmentation methods that are scalable to modern datasets.

This project used Livewire assisted segmentation in IMOD to manually create the small training datasets and CHM-LDNN and GLIA for large-scale automatic segmentation of the larger datasets. These programs were either created specifically or modified for the project along with several additional utilities to expediate the automatic processing.

3.2 Livewire: Assisted Segmentation

Livewire is an assisted segmentation method that lessens the time required to manually segment objects. It does this by predicting portions of the outlines and gives the user real-time feedback of the predictions so the user can immediately adjust for mistakes that it would make. There are several implementations of this in various image processing tools, however they all have serious drawbacks when segmenting EM images. Three open source versions are the “Intelligent Scissors” tool in GIMP (The GIMP Help Team, 2015), the “Ivussnakes” plugin for ImageJ (Baggio, 2006), and the “Live-Wire” function in the Insight Toolkit (ITK) (Tustison, Yushkevich, & Gee, 2008). The GIMP tool does not give real-time feedback but does allow adjustment of the results afterwards. The ImageJ plugin does provide real-time feedback but limits the user to short distances so this negates its benefits. The ITK function was never finished and no GUI for it exists publicly.

The Livewire algorithm is part of a class of algorithms called “Active Contours” and was originally developed in 1987 under various names including Intelligent Scissors (Mortensen & Barrett, 1995), Interactive Outlining (Daneels, et al., 1993), Live-Wire (Mortensen, Morse, & Barrett, 1992), and Snakes (Kass, Witkin, & Terzopoulos, 1987) each with various changes and adaptations to the concept. Prior to that time similar algorithms existed, however due to hardware limitations they were not ‘live’ and could not provide real-time feedback. The implementation of real-time feedback caused computational issues even into 2005 (Salah, Orman, & Bartz), but increases in hardware speed and advancements in the implementation make real-time feedback more achievable. The Livewire algorithm is composed of two major steps: filtering the image and then using a single-source shortest-path algorithm to connect two points. The filtering of the image reduces noise along with determining the weights that will be used to find the shortest path. Low values or black pixels representing the easiest pixels to “traverse” and high values or white representing the pixels that are hardest to “traverse”. Choosing the appropriate set of filters is crucial to getting usable results.

For this project, the Livewire method was adapted to work on very large, very noisy, grayscale EM images. It was implemented as a plugin to the IMOD segmentation software. The remainder of this section describes this specific implementation of Livewire as created for this project. All parts of it were chosen to meet the demanding conditions of working with EM images. As exposed in IMOD, this new Livewire tool has three tracing modes: center of a light region, center of a dark region, and boundaries between light and dark regions. The difference between these modes is simply in the filtering performed on the image. For tracing the center of a light or dark region, the image is first binned, reducing square sets of pixels to a single pixel, using median, mean, or Gaussian blurring. The default is to bin every 2x2 set of pixels with a median filter, but larger or noisier images may require larger bin sizes or other filters. The next step is to “accentuate” the image by applying a sigmoid logistic function with a half-maximum of middle gray (halfway between black and white), a steepness of 0.5, a minimum of black, and a maximum of white to each pixel. For an unsigned 8-bit grayscale image this equation becomes:

$$F(x,y) = \left\lfloor \frac{255}{1 + \exp(32/5 \; - Im(x,y)/20)} \right\rfloor$$

where $Im(x,y)$ is the value of the pixel at position x,y in the binned input image. This function causes any pixel that is closer to white than black to become whiter and any pixel that is closer to black than white to become blacker with values closer to middle gray being changed the most. The only difference between the modes for tracing in the middle of light and dark regions is whether the filtered image data is inverted or not (converting white to black and black to white linearly).

When the mode is set to the boundary between light and dark regions an edge-detection filter is applied instead of the accentuation filter. The IMOD version uses a modified Sobel-5 filter (Sobel & Feldman, 1968; Kekre & Gharge, 2010); for an unsigned 8-bit grayscale image it is defined as:

$$S_5 = \begin{bmatrix} -1 & -2 & 0 & 2 & 1 \\ -4 & -8 & 0 & 8 & 4 \\ -6 & -12 & 0 & 12 & 6 \\ -4 & -8 & 0 & 8 & 4 \\ -1 & -2 & 0 & 2 & 1 \end{bmatrix}$$

$$T = 170885700$$

$$G_x = Im * S_5 \qquad G_y = Im * S_5^T$$

$$G_m(x,y) = G_x(x,y)^2 + G_y(x,y)^2$$

$$F(x,y) = 255 - \left\lfloor \frac{255}{1 + \exp(25/4 - 50\, G_m(x,y)/T)} \right\rfloor$$

where Im is the binned input image, S_5 is the Sobel-5 kernel, T is the maximum value that $G_m(x,y)$ can have, G_x and G_y are the horizontal and vertical gradients, G_m is the unscaled, squared, magnitude of the gradient (squared L2-norm), and F is the final filtered image, which contains the inverted, sigmoid-accentuated, magnitude of the gradient. The $*$ symbol is the convolution operator. The gradients are inverted since normally an edge filter sets the most edge-like pixels to white and the least edge-like pixels to black. However, we want the most edge-like pixels to be black so they are easier to “traverse” with the shortest-path algorithm.

Internally the program supports Sobel-3, Sobel-5, Scharr-3 (Scharr, 2000), and Canny edge-detection filters (Canny, 1986), however Sobel-5 empirically gives the best results, so it is the only option presented to the user in IMOD to simplify the user interface.

Additionally, the program supports color images and other types of grayscale images. These are converted to unsigned 8-bit grayscale images internally by using the luminosity or other property of the of color pixels or shrinking the range of other grayscale image types.

Once the filtering is complete the next step is to find the shortest-path between two points, in terms of weights, within the image. Dijkstra's algorithm (Dijkstra, 1959) is the best option for this since, when implemented with a min-priority queue, it is the fastest known single-source shortest-path algorithm for arbitrary directed graphs with unbounded non-negative weights (Knuth, 1977). The filtered image can be thought of as a directed graph with each pixel in the image representing a vertex in the graph and each of those vertices having a directed edge to each of the neighboring pixels. The weights of the edges are simply given as the value of the pixel at the head of the edge, or $\sqrt{2}$ times the pixel value for diagonal edges. Since every pixel is the source and destination of 8 edges (except pixels along the edge of the image) and the weights can be easily obtained from the destination pixel value, the edges themselves do not need to be explicitly created.

Dijkstra's algorithm is a greedy algorithm that works iteratively, where every iteration determines the shortest path to another vertex (pixel) in the graph. The algorithm keeps 3 sets of vertices: processed, boundary, and unvisited. Initially the starting vertex is placed in the boundary set and all other vertices placed in the unvisited set. The processed set is initially empty. Also, every vertex is given a distance value which is initially infinite for all vertices except the starting vertex which is given a distance of 0. Finally, every vertex is also given a path direction that 'points' to the next vertex that is along the shortest-path to the starting vertex. For vertices in the processed set this value is valid. For vertices in the boundary set this value is may still be improved in further iterations. For vertices in the unvisited set this value is undefined. For the initial starting vertex, the value is special and there is no directional pointer as there are no previous points to traverse for the shortest path. During each iteration the following steps are taken:

1. Remove the vertex with the lowest distance value from the boundary set and add it to the processed set.
2. For each vertex connected to the vertex selected in #1:
    a) If the neighboring vertex is already processed, skip it.
    b) Otherwise calculate the distance d of going to the neighboring vertex through the selected vertex as the distance value of the selected vertex plus the weight of the edge to the neighboring vertex.
        i. If the neighboring vertex is part of the boundary set and d is less than the neighboring vertex's current distance value than the neighboring vertex's distance is set to d and the direction is updated to point to the selected vertex.
        ii. Otherwise, the neighboring vertex is part of the unvisited set and its distance value is set to d and its direction is set to point to the selected vertex along with being moved from the unvisited set to the boundary set.
3. If the boundary set is not empty, repeat from step #1.

Several iterations of this algorithm are diagrammed in Figure 3.1.

As long as the boundary set is implemented as a min-priority heap queue this is highly efficient, where priority is given as the distance value of a vertex. A simple binary min-priority queue allows for the minimum value to be removed and new values to be added to it in $O(\log(n))$ time where n is the current size of the boundary set. Taking advantage of the regular grid of vertices, specialized operations can be added to allow updating the distance values of vertices already in the boundary set in $O(\log(n))$ time as well instead of the typical $O(n\log(n))$ for binary heaps, thus greatly improving the time for step 2.b.i. in the algorithm above. Other forms of min-priority queues can be used, such as a Fibonacci queue (Fredman

& Tarjan, 1987) and may provide slightly better speed at the expense of more a complex algorithm and the possibility of not being able to take advantage of the regular grid of vertices which greatly increases the overall speed of the algorithm.

Various improvements were made to the Livewire program as part of this project to adapt it for typical EM datasets. One of the primary improvements is in the filters used, which are designed to deal with EM data. Additionally, several improvements were made so that very large datasets can be handled efficiently and in near real-time processing. For example, even while the algorithm is still running, the live wire is shown for any processed or boundary pixel back to the selected starting vertex as shown in the

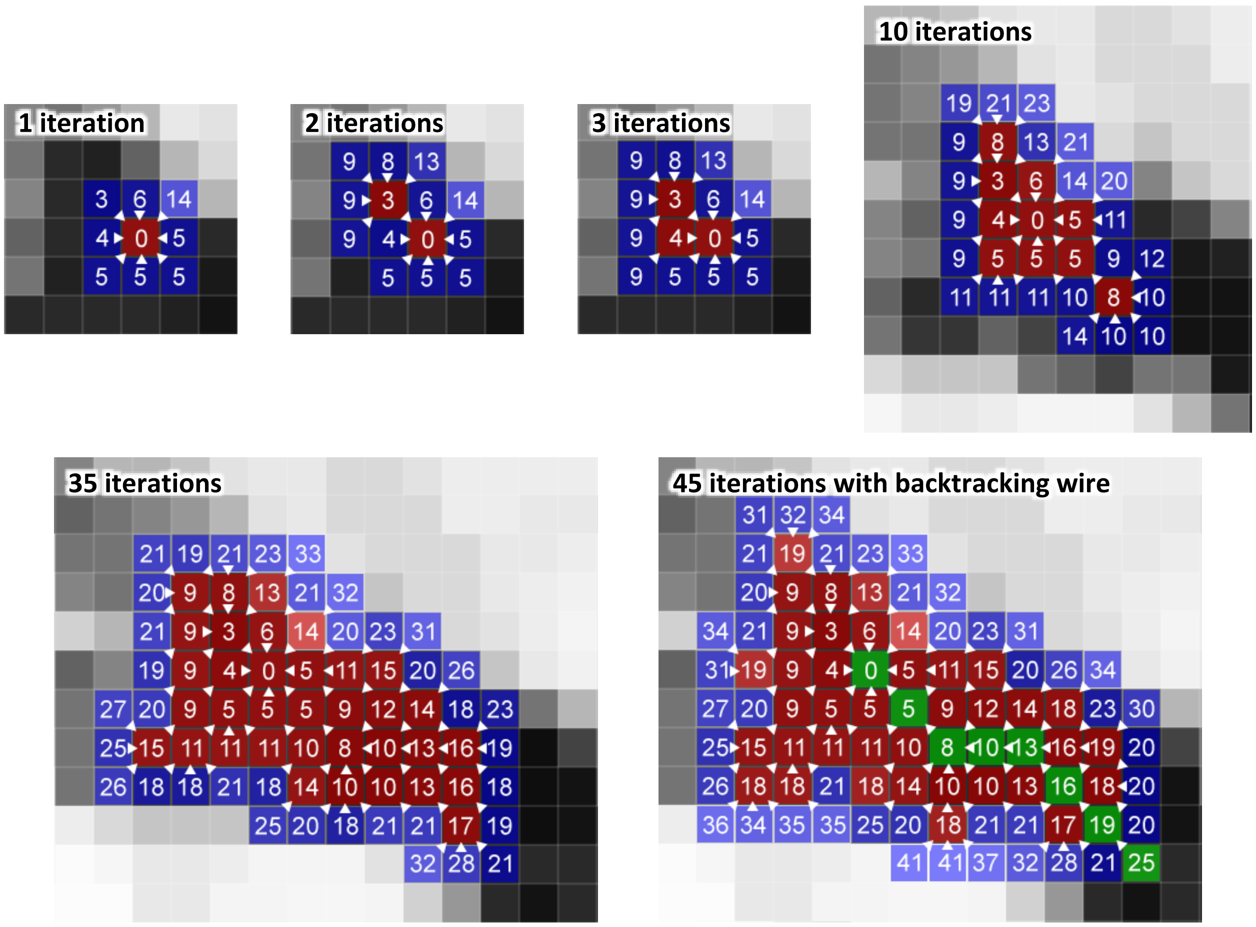

Figure 3.1. Livewire algorithm example at 1, 2, 3, 10, 35, and 45 iterations. Grayscale background is image data, with black having a value of 0 and whiter pixels have increasingly positive values. Red squares are part of the processed set, blue squares are part of the boundary set, and uncolored squares are part of the unvisited set. The numbers within each square indicate its distance to the starting vertex. The little white arrows on the edges of each box indicate the direction to travel back to get to the starting vertex. In the last image, the green squares indicate the backtracking wire from one of the computed pixels to the starting vertex following the arrows.

last panel of Figure 3.1. Since the algorithm generally works in expanding circles from the starting vertex, this means that pixels near the originally selected pixel in the image will be ready immediately and since the algorithm is quite efficient, even pixels further away are ready by the time the user can move the mouse further out, as shown in Figure 3.2. The initial filtering of the image and thus calculating the weights for each vertex takes the majority of the processing time. To improve responsiveness the weights are cached and reused between different selected pixels since they do not depend on which pixel is selected but only on the image data itself. However, with large images this could potentially be a prohibitively large

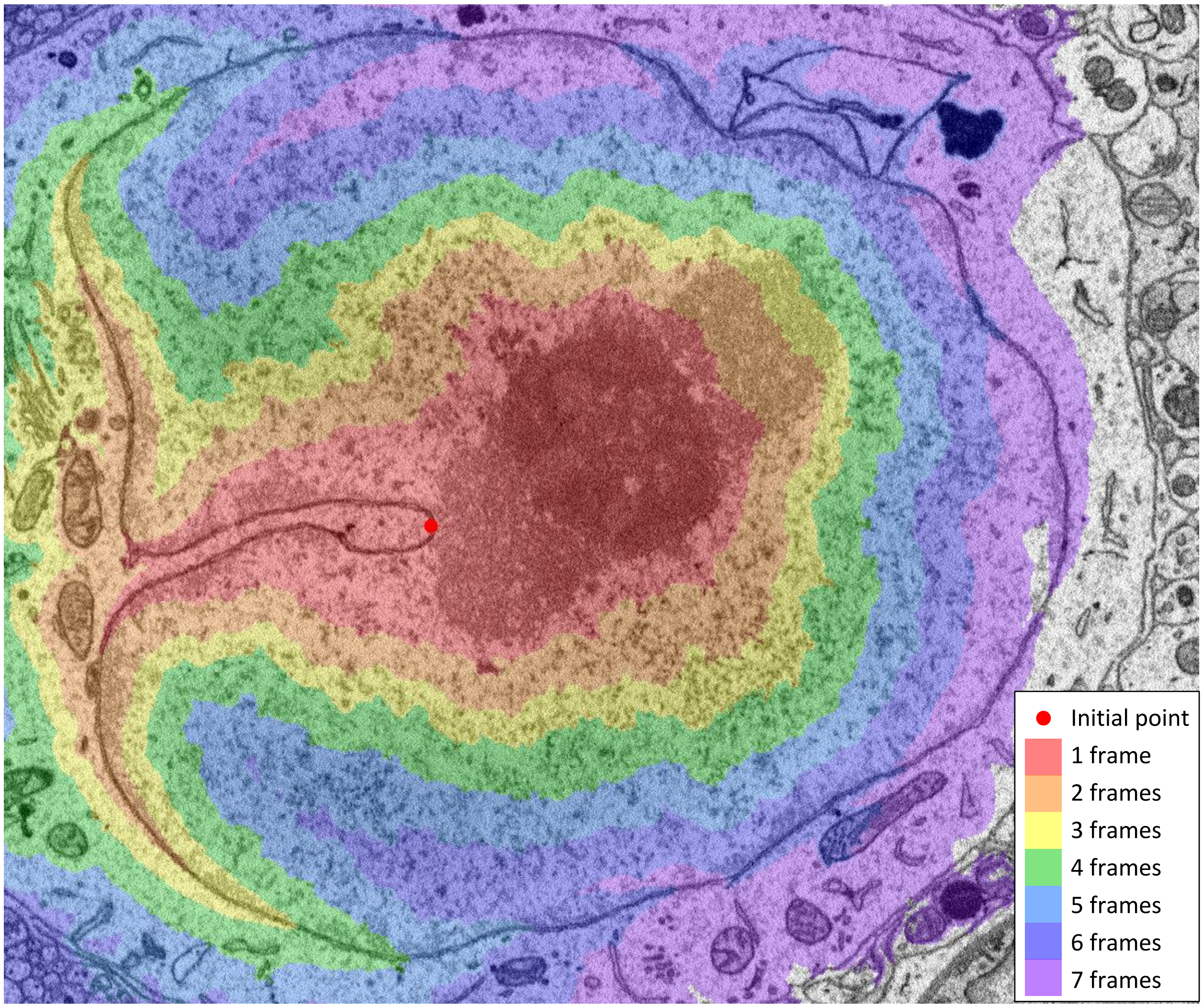

Figure 3.2. Livewire calculation speed. The red circle is the initial point clicked by the user. Each colored overlay region represents an additional video frame occurring where each frame is 1/60 of a second. Background is the image being processed and is 1001x1199 pixels. At this rate, in about 0.15 seconds all pixels on a standard monitor could be calculated and in less than half of a second an entire high-end 4K monitor could be calculated. This timing was performed on an Intel i7-4790 3.6 GHz processor (medium-high end processor).

amount of memory to store. Thus, to save on memory and processing time, for this implementation of Livewire the image is blocked into chunks and only chunks within a certain distance of the starting pixel is calculated.

One additional improvement implemented for this project within the IMOD version is that upon completion of a complete contour it is smoothed applying a few iterations of removing points from the contour where three consecutive points form a triangle below an area of 1 px^2 and moving points 25% closer to the halfway point between the point before and after it but only if the point is more than 0.2 pixels away from the halfway point. The contours are re-scaled so that the covered area is kept the same.

Overall, even on underpowered machines, the algorithm delivers nearly real-time processing over regions as large as 4096x4096 within images that are massive (tested with images up to 600 MP). The results of Livewire are not perfect, but since there is real-time feedback to the user, any mistakes it makes can be corrected before they are committed. This method has anecdotally been shown to improve manual segmentation speeds by 5-10x by various tracers from novices to experts for the segmentation of a wide range of objects including mitochondria, cell membranes, lysosomes, nuclei, and nucleoli. While this does still require user interaction, the time savings are quite sizable.

3.3 CHM-LDNN: Automatic Segmentation of Subcellular Structures

The Cascaded Hierarchical Model (CHM) was an algorithm originally developed at the Scientific Computing and Imaging Institute at the University of Utah by Seyedhosseini *et al* (2013; 2015) which uses information from a set of image processing filters to label the pixels within an image. It is an automatic image segmentation algorithm that uses supervised machine learning. Since it is supervised it requires a small subset of the dataset to be manually segmented to learn how the rest of the dataset should be automatically segmented and has two primary phases: training and testing. In some situations, the algorithm has been shown to be as accurate as humans in segmenting EM data. It performs best on the subcellular organelles such as mitochondria, lysosomes, nuclei, and nucleoli (Perez, et al., 2014). It is however very memory and computationally intensive and, as this project showed, it does not perform well on segmenting subcellular components such as endoplasmic reticulum (ER) and the cellular membrane itself.

3.3.1 CHM

CHM is hierarchical since it runs the classification process on several resolutions of the same image (termed the bottom-up step) and then a classification pass that incorporates the information from all resolutions (termed the top-down step). This allows the segmentation algorithm to detect large and small features alike that together indicate that a region should be classified as a particular subcellular component or not. The model is cascaded since this process can be repeated for greater refinement. However, empirically it has been shown that only a single bottom-up and top-down step is needed. In fact, the most recent publications refer to CHM as Contextual Hierarchical Model. The contextual part is that every pixel is classified by using a large, multi-resolution, contextual area.

The hierarchy is divided into levels with level 0 being the original image, level 1 being the image downsampled by half in each dimension (so a quarter of the size), and so forth. When the model is described as having 4 levels this indicates that there are actually levels 0 through 4 for a total of 5 different resolutions used in the bottom-up step. The cascaded nature of the models is described in terms of stages. A stage includes a top-down step and bottom-up step, except for the first stage which cannot include a top-down step and the last stage which does not have a bottom-up step. Thus, in the common case with 2 stages there is a single bottom-up step and top-down step.

There are 5 major components of CHM: downsampling, max pooling, upsampling, feature extraction, and classification. The first three components are needed to deal with the hierarchical and contextual nature of CHM while the final two are necessary components of any machine learning systems. The organization of these components for an example of 2 stages and 2 levels is shown in Figure 3.3.

The downsampling component is for shrinking the raw image data by half in each dimension when transitioning to a higher level. It uses bicubic interpolation (Keys, 1981) with anti-aliasing[7] after symmetrically padding the image to a multiple of 2 in each dimension. Each output pixel is the weighted average of pixels in a 4×4 neighborhood[8]. The max pooling component shrinks an image by half in each dimension by taking the max value of each 2×2 region of the image. Max pooling is used for

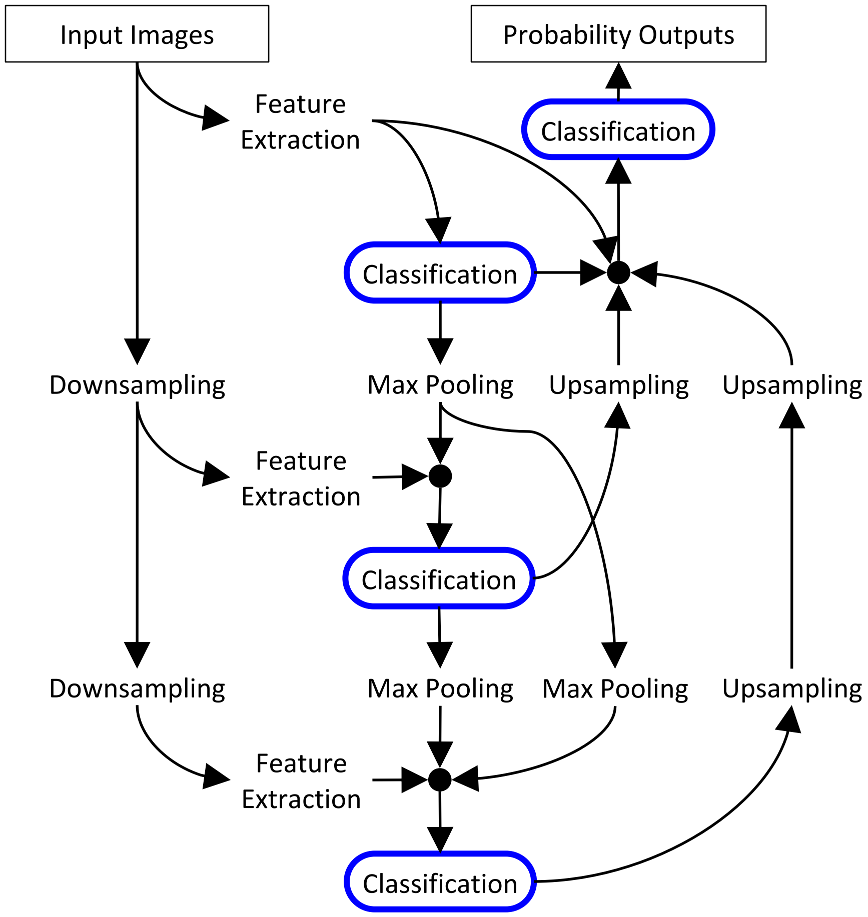

Figure 3.3. Contextual Hierarchical Model Illustration. Shows 2 stages ($S = 2$, width) and 2 levels ($L = 2$, height) but could be extended arbitrarily in either direction. The black circles indicate the combining of all the feature sources into a single set of features to feed into the classifiers.

[7] Anti-aliasing is a technique used to reduce the distortion artifacts when reducing the resolution of an image.
[8] The 2013 CHM paper says a mean 2×2 filter is used to downsample but that is not correct according to the actual code released.

downsampling the labels, which are binary images, and thus causes every output pixel to be true if any of the input pixels in the 2×2 region were true. Upsampling is the opposite of downsampling and increases the size of an image by two in each dimension using nearest neighbor interpolation, essentially replicating each pixel 4 times in a 2×2 pattern. This is used when transitioning to a new stage, and going back to level 0, since the higher level contextual information is smaller than the full-sized image.

Feature extraction uses a series of filters to generate the feature vector for each pixel based on the value of the pixel and its neighborhood. The classifier takes the feature vectors and classifies the pixels as being the object of interest or not. Before discussing these two parts in detail, we will look at how they fit into the overall CHM process.

In a very basic setup, if there was only a single stage, then the original image (or level 0 image) would be run through feature extraction and then the classifier, resulting in every pixel being classified. However, this would not utilize multiple resolutions of data and additional work is required to take this hierarchal information into account. We need at least 1 level and 2 stages to incorporate that information. In that scenario, after we have done feature extraction and classification on the level 0 image we downsample the image to create the level 1 image along with downsampling the results of the classification process on the level 0 image. Feature extraction is run on the new level 1 image. The downsampled classification results are used as features as well. All of these features (the features from using feature extraction on the level 1 image and the downsampled level 0 classification results) are used by the classifier to generate the level 1 classification results. This concludes the bottom-up step. The top-down step then uses the original resolution data again. It gathers features from feature extraction on the level 0 image, the level 0 classification results, and the up-sampled level 1 classification results and uses them all for classification. For a 2 level and 2 stage model see the diagram in Figure 3.3.

The model can use any set of filters for feature extraction and any classifier that is high-accuracy, fast, and robust against overfitting since it is used numerous times and on large datasets. The group at the University of Utah considered many pre-existing classifiers such as artificial neural networks, support vector machines, and random forests however they decided to introduce a new classifier that worked better with CHM which they called Logistic Disjunctive Normal Network (LDNN). The filters used for feature extraction in the original implementation were:

- **Haar-like features** implemented using Viola and Jones' (2001) 2-rectangle features (x and y directional features) of size 16. This detects horizontal or vertical bar-like features in the image, resulting in 2 features being extracted per pixel.
- **Histogram of oriented gradients (HOG)** from Dalal and Triggs (2005) with 9 unsigned orientation histogram bins per cell, a cell size of 8×8 pixels, and 2×2 cells per block. This detects edges in multiple directions normalized to their local areas resulting in 36 features being extracted.
- **Edge features** from the magnitude of the gradient using the first derivative of the Gaussian (Jain, Kasturi, & Schunck, 1995, pp. 168-169), grouping all edge directions into a single feature by using only the magnitude of the edge. The result is shifted to all possible places in a 7×7 square producing a total of 49 features[9].
- **Gabor features** using the work by Fogel and Sagi (1989) detect textures of varying orientation, frequency, and scale which mimic how some parts of the visual system work. The following parameters are used:
    - σ of 2, 3, 4, 5, and 6 which determines the spread of the filter
    - θ of $\frac{\pi}{6}, \frac{\pi}{3}, \frac{\pi}{2}, \frac{2\pi}{3}, \frac{5\pi}{6}, \pi, \frac{7\pi}{6}, \frac{4\pi}{3}, \frac{3\pi}{2}, \frac{5\pi}{3}, \frac{11\pi}{6}$, and 2π which dictates the orientation of the filter
    - $\lambda = \frac{2\pi}{W}$ of 2σ, 2.25σ, and 2.5σ which is the underlying sinusoidal carrier's wavelength
    - ψ of 0 and γ of 1 which creates a filter with the base phase and a square aspect ratio

[9] The 2013 CHM paper states that a Canny edge detector is used but that is not correct according to the actual code released.

resulting in 180 extracted features. The magnitude of the complex function is used, or this can be calculated as the L2-norm of the real functions with $\psi = 0$ and $\pi/2$. While this was the intent of the paper and code, due to a bug in the code the result of each of the fundamental filters are clipped to 0 and 1 in only 256 steps, meaning that the result of the feature is almost always 0, 1, or $\sqrt{2}$ after the L2-norm is performed.

- **Scale-Invariant Feature Transform (SIFT) Flow** features using the work by Liu *et al* (2011). These are based on densely sampled SIFT features by Lowe (1999; 2004). They are designed to support robust matching of objects while preserving spatial discontinuities. The parameters used are a cell size of 16×16 pixels, 4×4 cells per block, and 8 orientation bins resulting in a total of 128 features.

Additionally, the input image is sampled with a 21×21 sparse stencil for 81 features. This means that the raw image data of the pixel itself and all of the pixels up to 10 px away in each of the 8 directions: left, top-left, top, top-right, right, bottom-right, bottom, and bottom -left are included as separate features, Additionally, the max-pooled classification results are sampled with a 15×15 sparse stencil (up to 7 px away in each of the directions) for 57 features for every set of classification results used. Overall, this means that at least 476 features are extracted from the image along with 57 features for every set of prior classification results and fed to the classifier.

3.3.2 LDNN

LDNN is the classifier of choice for CHM and was designed explicitly for it. The goal of any classifier is to take a feature vector for a pixel and decide how likely it is that the pixel is part of a particular object or not. LDNN approaches this problem by treating the feature vectors as a point in a high-dimensional space (with 476 features extracted it is a point in 476-dimensional space). With the appropriate set of features, these points form clusters, forming one or more clusters that represent pixels which are not the object of interest and one or more clusters representing the object of interest. LDNN divides up the high-dimensional space to differentiate points that represent the object of interest from those that do not.

The LDNN classifier is similar to a previous classifier called a fuzzy min-max neural network (Simpson, 1992). Fuzzy min-max nets construct fuzzy (approximate) hypercubes around groups of points in the feature space. Each hypercube boxes up a set of points that either represents the object of interest or not. The quality of the classifier depends on the number of hypercubes constructed. The hypercubes must be axis-aligned however so in general a very large number of hypercubes are needed to isolate the points from each other. Additionally, fuzzy min-max nets are based on prototypes and an ad hoc training procedure is required. Instead, LDNN uses convex sets defined as the intersection of arbitrary half-spaces. This creates a difference: using arbitrary convex polygons to group items together in 2D space rather than axis-aligned rectangles. Polygons isolate the items much more cleanly than rectangles.

In a perfect system, the classifier would be a Boolean function, i.e. one that only ever produces true are false values, which answer the question: "Does the feature vector represent the object?" However, we will eventually never be completely sure and will instead produce a value that ranges from 0.0 if the pixel is definitely not the object and 1.0 is the pixel definitely is the object.

To create the definition for the LDNN, start with the definition of fuzzy min-max nets:

$$\tilde{f}(X) = \bigvee_{i} \left(\bigwedge_{j=1}^{n} h_{L_{ij}U_{ij}}(x_j) \right)$$

where $\tilde{f}: \mathbf{R}^n \to \mathbf{B}$ is the approximate classifier, $X \in \mathbf{R}^n$ is a feature vector with n as the number of features and x_j is the jth element, and h_{LU} is the hypercube function defined as:

$$h_{LU}(x) = \begin{cases} 1 & L \le x \le U \\ 0 & otherwise \end{cases}$$

Overall this uses a disjunction of conjunctions which is also known as the disjunctive normal form, for which any Boolean function can be written. However, since there is a finite number of hypercubes the classification is still approximate. As a first improvement, we can replace the hypercube function with a union of convex sets of half-spaces in $\mathbf{R}^n$:

$$h_{ij}(X) = \begin{cases} 1 & \left(\sum_{k=1}^{n} w_{ijk} x_k\right) + b_{ij} \geq 0 \\ 0 & otherwise \end{cases}$$

where w_{ijk} are the classifier weights and b_{ij} are the classifier biases, collectively forming the only parameters of the network. The next step is to make the classifier function $\tilde{f}$ differentiable. The conjunction $\bigwedge_j h_{ij}(X)$ can be replaced by $\prod_j h_{ij}(X)$ and using De Morgan's laws the disjunction $\bigvee_i q_i(X)$ can be replaced by $1 - \prod_i \big(1 - q_i(X)\big)$. Finally, the half-space function $h_{ij}(X)$ can be replaced with the logistic sigmoid function:

$$\sigma_{ij}(X) = \frac{1}{1 + e^{-\left(\sum_{k=1}^{n} w_{ijk} x_k\right) - b_{ij}}}$$

along with the differentiable disjunctive normal form approximation of the classifier as:

$$\tilde{f}(X) = 1 - \prod_i \left(1 - \prod_j \sigma_{ij}(X)\right)$$

Overall this model is described as an $N \times M$ LDNN model where N is the number of disjunctions and M is the number of conjunctions. The system finds N clusters of points in the feature space that represent the object and M clusters of points that do not.

To train the model the quadratic error of the classifier is reduced using a set of training examples $\mathbf{T}$ of pairs (X, y) where X is a feature vector and y is a binary label. The quadratic error is given as:

$$E(f, \mathbf{T}) = \sum_{(X,y)\in\mathbf{T}} \big(y - f(X)\big)^2$$

The error is reduced using the gradient descent method. This requires the derivative of the error, hence why there was a focus on creating a differentiable model. The only parameters are the weights w_{ijk} and biases b_{ij} so the derivative of the error is taken with respect to these values:

$$\begin{aligned}\frac{\partial E}{\partial w_{ijk}} &= -2\big(y - f(X)\big)\prod_{r\neq i}\big(1 - g_r(X)\big)\, g_i(X)\left(1 - \sigma_{ij}(X)\right) x_k \\ &= -2\big(y - f(X)\big)\frac{1 - f(X)}{1 - g_i(X)} g_i(X)\left(1 - \sigma_{ij}(X)\right) x_k\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial b_{ij}} &= -2(y - f(X)\prod_{r\neq i}\big(1 - g_r(X)\big)\, g_i(X)\left(1 - \sigma_{ij}(X)\right) \\ &= -2\big(y - f(X)\big)\frac{1 - f(X)}{1 - g_i(X)} g_i(X)\left(1 - \sigma_{ij}(X)\right)\end{aligned}$$

where $g_i(X) = \prod_j \sigma_{ij}(X)$ which are the conjunctions within the classifier. The gradient descent method can be used to minimize this equation. However, since there are $N \times M \times (n + 1)$ parameters, where n is the number of features which is at least 476 and there are typically going to be millions of training pairs the gradient descent method is very slow and will have issues with convergence. There are two approaches we can take to overcome this issue: stochastic gradient descent with mini-batches and gradient descent with dropout.

Stochastic gradient descent with mini-batches randomly groups the training samples into small subsets of training samples, updating the weight values after each mini-batch. The entire set of training samples is gone through several times, each time using different random groups of training samples (LeCun, Bottou, Orr, & Müller, 1998).

Gradient descent with dropout (Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 2012, pp. 5-8) uses a single training sample at a time but only works with half of the conjunctions and disjunctions at a time (one quarter of the total weights for a single sample). The samples are trained on in a random order and which weights are being updated are chosen randomly for each sample. The selective use of weights reduces overfitting and can increase the speed. Many iterations through all samples are performed. This method requires taking the square root of each of the products in the classifier function $\tilde{f}$ during the testing phase as follows to accommodate for the missing weights during training:

$$\tilde{f}(X) = 1 - \sqrt{\prod_i \left(1 - \sqrt{\prod_j \sigma_{ij}(X)}\right)}$$

In both situations the concept of momentum is used which means that the previous gradient information is used along with the current gradient information while performing the descent. This greatly increases the rate of convergence. CHM uses a stochastic variant of gradient descent, changing which samples are taken and in what order, to avoid local minima in the search for the global minimum. However, the sigmoid function σ_{ij} has horizontal asymptotes at 0 and 1 and thus if we targeted values of 0 and 1 for learning like one might expect for a binary classification problem the gradient descent would easily get stuck in local minima. Instead, targeting the values 0.1 and 0.9 are targeted which prevents saturation of the sigmoid function σ_{ij} (LeCun, Bottou, Orr, & Müller, 1998, pp. 10-12).

While this process refines the values of w_{ijk} reducing the error, an initial set of weights needs to be chosen to reduce the number of iterations the gradient descent method needs to achieve reasonable results. The initial values for the weights and biases can be logically setup using clustering of the data. For an $N \times M$ LDNN classifier we need $N + M$ clusters. The feature vectors of the training data $\mathbf{T}$ are partitioned into two sets corresponding to the binary value of y, all of the feature vectors X paired with a y of 1 are placed into $\mathbf{X}_+$ and all of the feature vectors X paired with a y of 0 are placed into $\mathbf{X}_-$. The $\mathbf{X}_+$ set is partitioned into N clusters with centroids $\mathbf{C}_+$ and the $\mathbf{X}_-$ set is partitioned into M clusters with centroids $\mathbf{C}_-$. This clustering is performed using the k-means algorithm. To be reasonably fast for large datasets only a subset (~1/10) of the training samples are used along with a multilevel variant of the k-means algorithm (Martin, Fowlkes, & Malik, 2002). The weight vectors $W_{ij} = [w_{ij1} \quad \cdots \quad w_{ijn}]$ are initialized as the unit vector from each C_{-j} to each C_{+i} which are the individual centroids from the negative and positive centroid sets $\mathbf{C}_-$ and $\mathbf{C}_+$ respectively. The bias terms are initialized such that

$\sigma_{ij}(X) = 0.5$ at the midpoints of the line connecting C_{-j} and C_{+i}. This is essentially defining a plane to separate each pairing of a negative and positive cluster from each other resulting in the positive clusters being isolated on all sides. The equations for calculating the initial weights and biases from the centroids are:

$$W_{ij} = \frac{C_{+i} - C_{-j}}{\left\| C_{+i} - C_{-j} \right\|} \qquad b_{ij} = -W_{ij} \cdot \frac{C_{+i} + C_{-j}}{2}$$

CHM uses different settings for the LDNN classifier based on the level and stage. At level 0, a 10×20 LDNN classifier using stochastic gradient descent with mini-batches of size 10 and a learning rate of 0.005 per sample is used. The number of iterations through the entire training set is either 15 for stage 1 or 6 for all other stages. For all other levels (levels 1 and higher) a 24×24 LDNN classifier using dropout gradient descent, 15 iterations through the entire training set, and a learning rate of 0.025 per sample is used. For all levels a momentum of 0.5 is used. A shortcut used throughout is that the biases b_{ij} are integrated into the weights W_{ij} as an extra 'feature'. This requires an extra feature to be added to each feature vector X that is always one. Thus $x_{n+1} = 1$, $w_{ij(n+1)} = b_{ij}$, and $W_{ij} = [w_{ij0} \quad \cdots \quad w_{ij(n+1)}]$. This allows the sigmoid equation to be simplified to the following:

$$\sigma_{ij}(X) = \frac{1}{1 + e^{-\left(\sum_{k=1}^{n} w_{ijk} x_k\right) - b_{ij}}} = \frac{1}{1 + e^{-W_{ij} \cdot X}}$$

which can be efficiently calculated using a single dot product using the BLAS library. Additionally, $\partial E / \partial w_{ijk}$ now works for biases as well which eliminates a separate equation for them. The initial weights and biases are still calculated separately as above.

3.3.3 Testing Improvements

CHM-LDNN is a very computationally and memory intensive process, especially when working with the massive images that are typical of EM datasets. The first improvement during the project was made to the original MATLAB implementation to parallelize the processing across multiple CPUs. This was implemented in order to greatly reduce the memory and time requirements for large images as is common with EM datasets. To accomplish this the program splits the images into several blocks while testing and processes each block separately in parallel using the `blockproc` method from the MATLAB Image Processing Toolbox. This made it possible to process large images. However, since several of the features are calculated using large neighborhoods around a pixel, processing in this manner creates boundary effects at the seams between every pair of blocks. To reduce this effect, each block was processed with a border coming from the neighboring blocks. Optimally the border size would be equal to $\hat{p} * 2^L$ where $\hat{p}$ is the maximum padding required for any filter and L is the highest-level number. The Gabor filter has the largest padding requirement of 18 pixels thus $\hat{p} = 18$. A typical value for L is 4. Thus, the optimal border size is 288 pixels. However, the typical block size is around 500×500 thus a border of that size is impossible in typical situations. Even for a much larger block size such as 1000×1000 this means that the majority of the data must be processed multiple times defeating most of the gains from block processing in the first place. Empirically, a border size of only 50 pixels is necessary in most cases and larger than that it does not reduce the edge effects much. However, for some situations a border size of 75 or 100 pixels improves the quality significantly, such as with segmentation of larger structures. Overall the block processing provides a significant reduction in memory usage and a modest increase in speed for a small drop in quality. The decrease in memory usage makes the testing process possible on very large images as only individual blocks are processed instead of the entire image at once like the original program attempted. However, the MATLAB `blockproc` method is designed for when each block can be processed rapidly which is not true for CHM and thus it is not optimized for this situation. The first few blocks are processed in serial instead of parallel to determine some of the properties of the output classification data meaning that if there are only a few blocks to be processed there is essentially no time gain and possibly some loss in speed. Overall, for large images this block processing is critical but does cause some reduction in quality. However, for smaller images this block processing hurts time and quality performance.

The MATLAB implementation had some serious bugs along with inefficiencies that made it difficult to use on large datasets and it had room for improvement in the accuracy. The entire algorithm was re-built using the Python programming language to create PyCHM. It uses the numerical and scientific libraries NumPy and SciPy (Walt, Colbert, Chris, & Varoquaux, 2011) and the optimizing static compiler Cython (Behnel, et al., 2011). NumPy and SciPy can use either the OpenBLAS (Qian, Yunquan, & Yi, 2013) or Intel MKL library (Intel Corporation, 2018) for their internal linear algebra computations with the Intel MKL being significantly faster although not open-source but still free to install and use. Other optional libraries are used if installed on the machine running PyCHM:

- pyfftw and the underlying FFTW C library is used to increase the efficiency of filters that depend on Fourier transforms, mainly the Gabor filter (Frigo & Johnson, 2005)
- Python Imaging Library (PIL or Pillow) to reading common image file formats such as PNG, JPEG, and TIFF (Clark & et al, 2017)
- h5py for reading HDF5-formatted MATLAB files for compatibility with the MATLAB version of CHM (Collette, 2013)
- psutil for determining the optimal number of threads to use.

All of these libraries are open-source and freely available to be run on nearly all systems which makes it significantly easier to run on any hardware that is available without needing an expensive license to run it on just a single machine. When running on a cluster this becomes even more important as every single node in the cluster needs a MATLAB license. Even though there is the ability to compile the MATLAB code so each machine does not need a separate license, the compilation license is incredibly expensive and compiled MATLAB programs are difficult to use with parallel processing. Additionally, moving it into the realm of Python allows other packages to utilize the functionality directly, such as embedding it into a server system for rapid visualization of results in a tile-based web display (Churas, et al., 2017).

For the testing phase of CHM the primary focus was improving the speed and memory usage of the filters since the bulk of the time was spent executing the filters. Minor improvements were also made to the evaluation of the classifier function itself including the use of a BLAS library for the main matrix multiplication so that it is highly efficient and multithreaded. Also, the HOG filter and the classifier for non-dropout models originally used 32-bit floating-point numbers but are now using 64-bit floating-point numbers in the Python version for increased accuracy and possibly speed on modern systems. Overall, when running the process on full-sized images the Python code runs approximately 6 times faster and utilizes almost half as much memory as the MATLAB code meaning that a testing run that once took a month to execute, now only take two and a half days with PyCHM using the same amount of resources. In addition, some of the fixes alleviated the minimum image size problem with the original MATLAB version, which required that all images used are least $15 * 2^L$ pixels in height and width, meaning that for a typical value of $L = 4$ the images had to be at least 240×240.

In the new Python version the memory usage is nearly optimal. The theoretical minimum amount of memory required to evaluate the classifier function is:

$$(N * M + M + n) * P * bytes\_per\_pixel$$

where P is the number of pixels in the block being evaluated and $bytes\_per\_pixel = 8$ for a 64-bit floating-point number. At level 0 the number of pixels P is maximized since at all other levels the image is downsampled. The maximal number of features n is at level 0 and stage ≥ 1 where there are $476 + 57 * (L + 1)$ features. Using a typical value of $L = 4$ for the highest-level number, the maximal number of features is $n = 761$. At this point CHM uses a 10×20 LDNN classifier and thus $N = 10$ and $M = 20$. With these values the minimum amount of memory is 7848 bytes per pixel. For a block with $P = 1{,}000{,}000$ pixels this means that the evaluation requires 7.31 GiB (7.85 GB) of memory. The Python code only uses an additional 60 MiB over this theoretical limit for various temporary values and status variables which is only an additional 0.8% memory usage. The MATLAB code in the same situation uses 14.4 GiB which is 97% more memory than is theoretically needed, nearly doubling all of the memory being used at the highest point.

The block processing system was also completely revamped when the program was rewritten in Python to improve quality, speed, and memory usage. First, each of the filters were adapted so that when given a block of image data they are able to read the contextual image data surrounding the block as far away as necessary for the filter without actually computing anything extra. In the original MATLAB implementation the filters assumed the blocks were surrounded by either all 0s or by a mirror of the data in the block itself. With this improvement, the full image can now be broken into blocks without requiring a border or overlap to overcome edge effects. This means that the image data is not completely filtered multiple times in the border regions. Previously with 500×500 blocks and 50-pixel borders 36% of the work was being wasted ($1 - \frac{400*400}{500*500}$), with loss of accuracy at the edges of each of block. Breaking the image into blocks still does some additional work compared to working on the image all at once since many of the filters are able to conserve some of the computations as the work across an image, however if the image were not broken into blocks significantly more memory would be required greatly limiting the hardware the algorithm can be run on.

One downside to this method is that each level needs to be completely filtered and evaluated before proceeding to the next level. With MATLAB's `blockproc` method each block was completely processed through all stages and levels before starting the next block. The revamped version processes a single stage/level across all blocks before moving onto the next stage/level. The drawback here is that the program must store the complete results of each level within a single stage overall requiring at most $26.6\overline{6}$

bytes per pixel[10] in the entire image not including the per-block evaluation memory. However, this memory is memory-mapped and can be quickly swapped into and out of main memory to secondary storage by the system since only small amounts of the memory are used at any given time.

One of the primary ways that CHM testing is used is for it to only process a subset of the blocks in a single run. This allows a single very large image to be processed by several different machines in parallel with the results merged together in the end. In the MATLAB block processing system it was trivial to only process some of the blocks since each block was performed separately. The Python version uses approximately 12 times less resources so this feature is not as necessary as it was for the MATLAB CHM, it is still desirable for extremely large images or integration with other tiling systems. Its implementation becomes much more of a challenge in Python however, since blocks depend on the computation of neighboring blocks for contextual information since bordering regions are no longer used. The use of only a subset of blocks is supported in Python and does save considerable memory and computational time compared to working with the entire image, but unlike with the MATLAB CHM, the savings are not as noticeable and do not scale linearly with the number of blocks being tested. The Python CHM attempts to work on groups of blocks to reduce the border of those blocks thus reducing the additional number of blocks that need to be processed. Thus, instead of specifying individual blocks for processing, the user specifies the total number of groups of blocks the image will be processed within and which group it is. For example, if you were spreading out the work for a single image across 10 nodes in a cluster, each process would be instructed that it is part of a group of 10 and what number within the group it is. Then the set of tiles is determined in order to reduce the amount of extra work needed by any node and to

[10] This value comes from the extra room needed to store the entire original image, the classifier results for each level ($\sum_{i=0}^{L} 1/2^i$ values per pixel which is $4/3 = 1.3\overline{3}$ as $L \to \infty$ or ~ 10.656 for $L = 4$), and room to store temporary data including downsampled images and classifier results. Summing those we get $1 + 1.3\overline{3} + 1 = 3.3\overline{3}$ values per pixel. Each value is stored as a 64-bit floating-point number resulting in $8 * 3.3\overline{3} = 26.6\overline{6}$ bytes per pixel. This does not include the rounded-up dimensions during downsampling however that adds a very small amount as long as the image is large and roughly square.

make sure each node has roughly the same amount of work to perform. The aim is to have all the tasks finish near the same time (for the algorithm used see Appendix A). However, since the process now requires approximately one twelfth of the overall resources, dividing a single image across multiple nodes is less necessary.

The Python version of CHM-testing produces classification results that are extremely close to those that are produced by the MATLAB-CHM testing code when running on the models created by the MATLAB training code. The main differences are near the boundaries of the images and seams between blocks. The seams are due to insufficient overlap being used in the MATLAB version while there is essentially unlimited overlap in the Python version. The differences along the boundaries of the entire image are primarily due to differences in how the Python and MATLAB codes handle padding of the entire image to provide data to filters that need points outside of the image – in several cases the MATLAB code assumes the image is surrounded by 0s (pure black) instead of something more accurate such as mirroring the image data itself. When the images are surrounded by 0s, filters like edge detection always see a very large edge at the boundary of the image. The Python code attempts to reproduce this poor behavior when using MATLAB-created models but it prioritizes increased efficiency and code abstraction over perfect reproduction.

3.3.4 Training Improvements

While the Python implementation of the testing phase of CHM showed a great improvement for speed and memory usage there was no significant improvement in accuracy except along the edges of the image and at the seams between blocks. Training also needed to be re-created in Python to improve accuracy as changes in the filters also change the necessary weights for the LDNN model. The implementation of the filters in PyCHM fixed several mistakes made in the original implementation of CHM. The filters were modified as follows:

- **HOG**: the original implementation had an implicit padding of 0 around each 16×16 block causing highly inaccurate output. Additionally, the blocks were misaligned and values were assigned to histogram bins in a non-standard way. The re-implemented method made it more accurate to the original description of the algorithm by Dalal and Triggs (2005). Additionally, it is significantly faster (30-35x faster) at the expense of using significantly more memory, but the additional memory is less than the extra memory required by other filters so does not increase the overall memory usage of the program at all.
- **Edge:** the original implementation padded the image with 0s creating large edge signals at the boundary of the image; this was replaced with symmetric padding which greatly reduces this source of error along the boundary.
- **Gabor:** the original implementation clipped the result of the convolutions with the Gabor kernels to be from 0 to 1 with 8 bits of resolution even though these intermediate results have a mean of 0 and the larger kernels produced data mainly outside of the -1 to 1 range. In the end, the Gabor filter was typically only generating the values 0, 1, and $\sqrt{2}$ due to this bug and never generating values outside of the range $[0, \sqrt{2}]$. However, modifying the implementation to not perform clipping resulted in several issues as well. One issue was the squaring of the negative values which caused all values to become positive and thus there was no distinction between the filter and its 180° analog, losing a lot of information that the Gabor filters are designed to supply. Several solutions to this problem were attempted:
    - No clipping at all, ignoring the loss of information caused by the duplicated orientations
    - Set all negative values to 0 and saving the magnitude, which is the original CHM method except without clipping the upper bound to 1 and keeping the full resolution of the values

    - Correcting for the non-zero DC offset of Gabor filters but otherwise the same as the first attempted solution (Movellan, 2008)
    - Keep the real part of the filter instead of calculating the magnitude of the complex value.

  Ultimately none of these solutions changed the results significantly as compared to not using the Gabor filter at all and the results were dependent on what structure was being classified. This, combined with the fact that the Gabor filter is by far the most expensive filter to calculate of all of the filters used, in terms of both memory and time, in most cases it was simply not used. When used, for example with mitochondria, it was used as the original CHM had created it.
- **Haar:** minor changes were made for efficiency that ended up producing slightly different outputs (on the order of 10^{-8}) due to a different set of floating-point rounding errors occurring.

Additionally, the Frangi filter (1998) was added becauseit robustly detects tube-like structures such as cell membranes and endoplasmic reticulum. The parameter β is set to 0.5 and c dynamically set to half of the maximum Frobenius norm of the Hessian matrices calculated during the process as per the recommendations in the paper. The σ parameter runs over the values 2, 3, 4, 5, 7, 9, and 11 for both the original and inverted image so that both black and white tube-like structures of different sizes can be detected. This results in a total of 14 extracted features which increases the total number of features to 490 features plus contextual features.

3.3.4.1 Normalization of Filters

Prior to fixing these filters most of their outputs were at least close to the range 0.0 to 1.0. This is important since the LDNN algorithm greatly favors features with larger values than features with smaller values. This means that features with a very small range compared to the other features will essentially be ignored while ones with a large range will dominate the learning process. The most prominent example of this was seen when the Gabor filter was corrected. Previously it only generated values from 0 to $\sqrt{2}$

(~ 1.414). Once it was corrected the narrower Gabor filters generated values from 0 to 2 while wider Gabor filters can generate values from 0 to 45. Even same-size Gabor kernels that just have a different orientation generate a different range of values. Due to this the Python version of CHM initially failed at being able to classify just about anything correctly since it was only paying attention to the largest Gabor filters and essentially ignoring all other filters (even non-Gabor filters). The later discovery that the Gabor filters were not beneficial in most cases made it even more obvious that just focusing on the largest Gabor filters would always lead to poor results.

Ideally, of course, a wider filter or a filter with a different orientation should not be considered more important than another filter before the learning process begins. Rather, within LDNN it should start off by being able to consider all the features equally and then work on changing their relative weights. When LDNN starts with unequally scaled data it can never find the appropriate weights to "un-scale" the data. The change in influence is due to issues in the definitions of the half-spaces and calculating the initial weights and biases. The initial weights and biases are calculated by placing the planes defining the half-spaces half-way between a cluster of negative and a cluster positive features vectors. If one feature axis is scaled disproportionately relative to the other features, then the plane will be closer to perpendicular to that axis. It becomes closer to perpendicular as the scale becomes larger, and as it becomes more perpendicular there is less variability in the other features represented by the plane.

An example can be worked out in 2D with just a single negative and positive cluster to illustrate the scaling issues with LDNN, as shown in Figure 3.4. The negative cluster is at $(0,0)$ and the positive cluster is at $(1,1)$. The line separating the two clusters is $x_1 + x_2 = 1$ resulting in $w_1 = w_2 = 1$ and $b = -1$. If the x_1 feature is now scaled by 2 then the positive cluster moves to $(2,1)$ with the separating line now at $2x_1 + x_2 = 5/2$ with the values $w_1 = 2$, $w_2 = 1$, and $b = -5/2$. Some points will have changed which side of the line they are on; for example $(0,1.5)$ will have been on the positive side and now it is on the negative side after scaling. The scaling of x_1 has reduced the influence of x_2 on the outcome of the result. This effect does not only effect calculation of initial weights but is present throughout the learning process.

The solution is to normalize the features before learning so that they have similar distributions to each other. There are several possible ways to normalize the data and the following methods are implemented in the Python CHM training program:

- **minimum/maximum** (min-max): linearly scale and shift each feature so that its minimum value is 0.0 and the maximum value is 1.0. All training data will be between 0.0 and 1.0 however outliers in a feature will skew the data significantly.
- **mean/standard deviation** (mean-std): shift the data so that the mean is at 0.5 and scale the data so that ± 2.5 standard deviations lie at 0.0 and 1.0. Even though $\sim 5\%$ of the training data will end up outside of the 0.0 to 1.0 range the effect of outliers will be greatly reduced.
- **median/median absolute deviation** (median-mad): shift the data so that the median is at 0.5 and scale the data so that ± 2.5 standardized median absolute deviations (MADs) lie at 0.0 and 1.0. The median absolute deviation is the median correlate of standard deviation. The "standardized" MAD is the MAD divided by ~ 0.6745 ($\Phi^{-1}(3/4)$ where Φ^{-1} is the inverse cumulative distribution

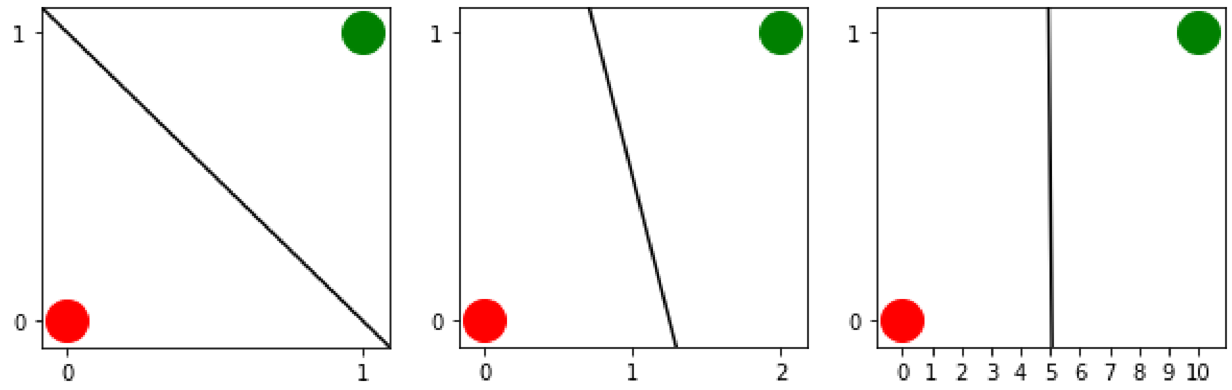

Figure 3.4. Data scaling issues with LDNN. The green dots (upper-right) represent the positive clusters and the red dots (lower-left) represent the negative clusters. When the axes are equally scaled (left plot) the line that divides them is $x_1 + x_2 = 1$. As the x_1 axis (horizontal axis) is scaled by 2x (middle plot) and 10x (right plot) the dividing line becomes $2x_1 + x_2 = 5/2$ and $10x_1 + x_2 = 101/2$ respectively. The dividing line becomes significantly perpendicular to the x_1 axis and parallel to the x_2 axis, thus reducing the ability for x_2 to influence the outcome.

function or percent-point function of the normal distribution) so that it becomes a consistent estimator with standard deviation. Like with mean-std some of the training data will end up outside of the 0.0 to 1.0 range but this has even better tolerance to outliers since median and MAD are both more robust in the face of outliers than mean and standard deviation are. This is the default setting in the program and is commonly used in similar statistical analyses.

- **interquartile range** (iqr): linearly scale and shift each feature so that the value $Q_1 - 1.5IQR$ becomes 0.0 and the value $Q_3 + 1.5IQR$ becomes 1.0 where Q_1 is the first quartile of the data (the value which has 25% of the data below it), Q_3 is the third quartile (the value which has 75% of the data below it), and $IQR = Q_3 - Q_1$ is the interquartile range. This means the range from 0.0 to 1.0 contains a $4IQR$ range from the original data. This is the most robust against outliers but the amount of data outside of the 0.0 to 0.1 is much less predictable than the other methods.

The information on how to scale and shift the data is calculated per-feature and saved so that the same transformations can be applied during testing so that their values are consistent. The mean-std and median-mad normalization methods expect the data to be roughly normally distributed and do not work for some features such as Frangi which has a highly bimodal distribution. These filters use a feature-defined normalization method instead.

The normalization methods described above are only applied to the data while it is being used during the gradient descent process. Additional changes were made to CHM to include a separate normalization process during k-means clustering as well which is used to calculate the initial weights. k-means clustering works best with “whitened” or “sphered” data (Coates & Ng, 2012) however this was not done in the original CHM implementation leading towards bias in the larger features when calculating the initial weights. Whitened data is when the variance of each feature is equal to 1 but the mean can be anything. This is accomplished by dividing each feature’s data by its standard deviation. This is a different requirement than for normalizing the data going into the gradient descent portion of LDNN which works best when all the data is in the range 0.0 to 1.0 instead of just having a fixed variance for each feature. Thus, there are multiple normalization steps when training a LDNN classifier.

3.3.4.2 Gradient Descent Improvements

Besides improvements with the quality, speed, and memory usage of the filters, the training process itself was improved. The speed of the stochastic gradient descent method has been greatly increased by vectorizing and parallelizing the calculation of the gradient and error across an entire mini-batch along with using 64-bit floats instead of 32-bit floats. In both gradient descent methods in the original implementation some of the temporary memory that is allocated is never freed up, leaking memory which contributed to its overall high memory usage during the training process. The method used to shuffle the data randomly for sample and feature selection was biased due to using a non-robust random-number-generator. On some systems using more than 32,767 samples (a single 180x180 image exceeds this) results in completely undefined results. Both of these issues have been addressed with the elimination of memory leaks and replacing the random number generator with RandomKit (Roy, 2005) which implements the robust Mersenne Twister PRNG (Nishimura & Matsumoto, 1998). RandomKit is included with the NumPy library and does not add an additional dependency.

3.3.4.3 Image Masking

An additional feature added to the training process is the support for masking of the training data. The MATLAB version of CHM requires that the training data has every single pixel classified as either not part of the object or part of the object. This means that only rectangular regions are permitted and there will inevitably be edge effects as no outside image data can be given. With masking, an additional set of binary images is provided that describe which pixels have a valid classification, every white (1-value) pixel in the mask is a pixel with a valid classification in the training data. This allows for non-rectangular regions to be used along with making sure that a wider context is provided to reduce edge effects. Masking also allows for a “progressive” segmentation. This progression is done by first segmenting an object and then masking that object out when training another object. This is repeated for each object. This can increase quality in situations where some objects are easily mistaken by the classifier as another object which is more reliably segmented. Internally, the features are still generated for the entire input data but only pixels which are not masked out are considered during the learning process of LDNN.

3.3.5 Post-Processing: Thresholding

The output of testing with CHM is an image with values that range from black to white (either 0 to 255 for 8-bit grayscale images or 0.0 to 1.0 for floating-point grayscale images). These values indicate if a pixel is either not the object (closer to 0.1 or mostly black), is the object (closer to 0.9 or mostly white), or somewhere in between (e.g. 0.5 or gray) if it is unsure how to classify the object. The output of CHM is typically values close to 0.1 or close to 0.9 with very few gray values. The reason 0.1 and 0.9 are the most common values is that the LDNN algorithm targets the values 0.1 and 0.9 instead of 0 and 1 to prevent oversaturation of the sigmoid function.

However, what we really want is a purely binary classification of whether the pixel is the object we trained against or not. The data must be thresholded to convert from the grayscale image to a binary (black and white) image. In quick tests, a value of 0.5 (50% gray) is frequently chosen as the threshold, with all values whiter than this being set to pure-white and all values blacker than this being set to pure-black. However, this method is a bit arbitrary and may introduce a small amount of noise.

Instead, our pipeline uses Otsu’s method to automatically determine a suitable gray value at which to threshold the image (Otsu, 1979). Otsu’s method is a clustering-based method that calculates the optimum threshold value that separates all pixel values into two classes which minimizes the intra-class variance. In other terms, this searches through all threshold values to minimize the value of:

$$\sigma_\omega^2 = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t)$$

where ω_0 and ω_1 are the probabilities of the two classes separated by a threshold t and σ_0^2 and σ_1^2 are the variances of these two classes. The class weights/probabilities can be computed from a histogram with L bins (typically 256) and a p_i proportion of all pixels lie within the i-th bin as follows:

$$\omega_0(t) = \sum_{i=0}^{t-1} p_i \qquad \omega_1(t) = \sum_{i=t}^{L-1} p_i$$

which means that $\omega_0 + \omega_1 = \sum_{i=0}^{L-1} p_i = 1$. The threshold t must be $0 < t < L$. Otsu showed that minimizing the intra-class variances is equivalent to maximizing the inter-class variances which then avoids calculating the variance of each class:

$$\sigma_b^2(t) = \sigma^2 - \sigma_\omega^2(t) = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2 = \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2$$

where μ_0 and μ_1 are the class means and μ_T is the total mean level of the original image. They are defined as:

$$\mu_0(t) = \sum_{i=0}^{t-1} i\frac{p_i}{\omega_0(t)} \qquad \mu_1(t) = \sum_{i=t}^{L-1} i\frac{p_i}{\omega_1(t)} \qquad \mu_T = \sum_{i=0}^{L-1} ip_i$$

which means that $\omega_0\mu_0 + \omega_1\mu_1 = \mu_T$. This formulation can lead to very fast computations on modern machines for the entire dataset at once by first calculating the cumulative moments of the histogram:

$$\omega(t) = \sum_{i=0}^{t-1} p_i \qquad \mu(t) = \sum_{i=0}^{t-1} ip_i$$

The other values can now be derived in terms of these cumulative functions:

$$\begin{array}{ccc} & \omega_0(t) = \omega(t) & \omega_1(t) = 1 - \omega(t) \\ \mu_T = \mu(L)/\omega(L) & \mu_0(t) = \mu(t)/\omega(t) & \mu_1(t) = \dfrac{\mu_T - \mu(t)}{1 - \omega(t)} \end{array}$$

Resulting in a final equation of:

$$\sigma_b^2(t) = \frac{\big(\mu(t) - \mu_T\omega(t)\big)^2}{\omega(t)\big(1 - \omega(t)\big)}$$

Finding the max of this function is a simple linear search. Once the optimal threshold value is found the image can be converted into a binary image. Otsu's method works best for bimodal distributions with a deep and sharp valley separating the two peaks. This is mostly true for the probability map images generated by CHM and PyCHM, however the separation between the two peaks is not sharp and thus the gray values in the middle may still be misclassified.

One way to improve upon this is to take into account the context of each pixel. If a gray pixel is surrounded by all white pixels it should likely be classified as white. Likewise, if a gray pixel is surrounded by all black pixels it should likely be classified as black. This can be accomplished with hysteresis thresholding in addition to multi-level Otsu's method.

Multi-level Otsu's method is an extension of Otsu's method in which two or more threshold levels are determined instead of a single level (Liao, Chen, & Chung, 2001). This means that instead of just creating a binary image (black and white), a ternary image (black, gray, and white) or an image with even more levels can be created.

The inter-class variance function $\sigma_b^2(t)$ can be generalized to M levels as follows:

$$\sigma_b^2(t_1, t_2, \dots, t_{M-1}) = \sum_{k=1}^{M} \omega_k (\mu_k - \mu_T)^2 \text{ where } 0 < t_1 < \cdots < t_{M-1} < L$$

with:

$$\omega_k = \omega(k) = \sum_{i \in C_k} p_i \qquad \mu_k = \frac{\mu(k)}{\omega(k)} = \sum_{i \in C_k} \frac{i p_i}{\omega(k)}$$

$$C_k = [t_{k-1} + 1, \dots, t_k] \text{ with } t_0 = 0 \text{ and } t_M = L$$

By noting that $\sum_{k=1}^{M} \omega(k) = 1$ and $\sum_{k=1}^{M} \mu(k) = \mu_T$ the inter-class variance equation can be rewritten as:

$$\sigma_b^2(t_1, t_2, \dots, t_{M-1}) = \sum_{k=1}^{M} \omega(k) \mu^2(k) - \mu_T^2$$

The final term $-\mu_T^2$ is constant across any choice of threshold values so it does not influence the calculation of the maximum and thus the equation that actually needs to be computed is only:

$$\sigma_b'^2(t_1, t_2, \dots, t_{M-1}) = \sum_{k=1}^{M} \omega(k)\mu^2(k)$$

For 2 thresholds (3 levels or $M = 3$) this can be very efficiently calculated using an $L \times L$ matrix as long as L is relatively small (a typical value is 256 which means less than a megabyte is needed for such a matrix). In fact, only the upper-right triangular region needs to be calculated since $0 < t_1 < t_2 < L$ cutting the computation and memory down by more than half. Once every value has been calculated finding the maximum is trivial.

Once the ternary image is created using the thresholds calculated using a 3-level Otsu's method, hysteresis thresholding can be used to convert it into a binary image. In hysteresis thresholding any black pixel will remain black and any white pixel will remain white. This leaves the gray pixels to be separated into the black and white classifications. This is done by taking the neighbor of every white pixel that is gray and classifying it as white. Any pixels newly classified as white repeat this process as well. In terms of mathematical morphological operations this is a binary dilation operation. The structuring element used has connectivity 1 meaning that the neighbors of a pixel include the pixels directly above, below, left, and right of the pixel but not the diagonals. In other situations, a connectivity of 2 can be used which includes the diagonals as well which more aggressively assigns gray pixels into the white classification.

3.3.6 Pre-Processing: Histogram Equalization

Raw images produced by SEM frequently have low contrast meaning that the difference between a light pixel and a dark pixel is not very high compared to the range of available values. Histogram equalization can be used to increase the contrast of the image without changing the relative ordering of the pixel values by applying a linear transformation to each pixel. The linear transform is derived from converting the histogram of the image to a desired histogram, typically a histogram with all equal-probability bins.

Standard histogram equalization is typically used for visualization by humans on monitors. However, it is not appropriate for images that will be processed by machines, even though most image processing toolkits only include the standard histogram equalization. It is inappropriate because it is unable to get the image histogram to completely match the target histogram except with very contrived examples and causes a loss of information in the image since some bins in the original histogram will be combined but no histogram bins will be split in the attempt to match the target histogram. Previously, Perez *et al* started using a newer method called exact histogram equalization (2014).

Several exact histogram equalization methods have been developed. These assign a strict ordering to all pixel values and then are able to match the target histogram nearly identically. This does preserve overall ordering of pixels (lighter pixels will remain lighter than darker pixels) however pixels that had the same value initially may end up with two different values based on their strict ordering.

The strict ordering of the pixels can be assigned in various ways. The method proposed by Coltuc *et al* (1999; 2006) uses the mean of the local neighborhood of each pixel at expanding distances to determine the order. If two pixels have the same value, then the means of the four neighboring pixels are used to distinguish them. If that value is the same then the means of the eight neighboring pixels are used. This process is continued up to an order of 6 which includes all pixels in a 5×5 area around each pixel. This method for calculating the strict ordering of the pixels is extremely time and memory consuming.

An alternative method for determining a strict ordering was developed by Nikolova *et al* (2013; 2014) which is faster than the local-means method and believed to have superior performance. Their approach, which is a variational approach, attempts to reconstruct the original real-valued image from the discretized image. The concept is that the image was acquired from a continuous source and then discretized when it was digitized. However, we can estimate what the original, physical, image was and thus assign a unique value to each pixel, allowing us to find a strict ordering of all of the pixels.

Their method minimizes the ℓ_1-norm of the total variation of the image. However, this would normally be incredibly computationally expensive, but with an appropriate selection of a smooth-approximation of the ℓ_1-norm this becomes significantly easier to compute. They required that the approximation had an explicit derivative and inverse of the derivative and that those forms were easily computable (most important they lacked a square root which is a very expensive operation). Their search resulted in the following ℓ_1-norm approximation:

$$\theta(t, \alpha) = |t| + \alpha \log\left(1 + \frac{|t|}{\alpha}\right)$$

where t is the difference between two values and α is the factor of the approximation, with $\alpha = 0$ being exactly the ℓ_1-norm. This function has the following derivative and inverse of the derivative:

$$\theta'(t, \alpha) = \frac{d}{dt}\theta(t, \alpha) = \frac{t}{\alpha + |t|} \qquad \xi(t, \alpha) = \theta'^{-1}(t, \alpha) = \frac{\alpha t}{1 - |t|}$$

which each only have a single division in them and thus are fairly easy to compute.

Given an image, called f, the real-valued version of the image, called u, is initially set equal to the original image, thus $u = f$. The minimization process then updates u during every iteration until every value in u is unique. This minimization is performed by first calculating the sum of the slopes (derivative) of the ℓ_1-norm of every pixel relative to each of its four neighbors, generating an image-sized matrix D where every (x, y) element in the matrix is calculated as follows:

$$\begin{aligned} D(x, y) = {} & \theta'(u(x, y) - u(x, y - 1), \alpha_2) + \theta'(u(x, y) - u(x, y + 1), \alpha_2) \\ & + \theta'(u(x, y) + u(x - 1, y), \alpha_2) + \theta'(u(x, y) - u(x + 1, y), \alpha_2) \end{aligned}$$

where $u(x, y)$ is a pixel in the current approximation of the real-valued image. Any term of this formula that requires data outside of the image (i.e. $u(0, -1)$) is simply dropped. Once the slopes are calculated, then the current approximation of the real-valued image is update as follows:

$$u = f - \xi(\beta D, \alpha_1)$$

where β is a constant describing the speed of convergence. This process is then repeated with the new values in u. Originally, they determined that at most 35 iterations of this process were required to achieve uniqueness (Nikolova, Wen, & Raymond, 2013), however upon further examination realized that only 3 to 5 iterations were required in all situations to acquire good-enough results (Nikolova & Steidl, 2014). The reduction in the number of iterations combined with a more efficiently computed ℓ_1-norm resulted in a 30 to 50 times acceleration of the computation making this the fastest method for exact histogram equalization when published (Nikolova & Steidl, 2014). It was also determined that, while the process could be cut short as soon as the uniqueness property held, calculating uniqueness of values was significantly more expensive than simply performing additional iterations (each iteration is an $O(n)$ operation while uniqueness-checking is an $O(n \log n)$ operation). When implementing this, the number of iterations was user-selectable but defaulted to 5 and the constants were fixed as suggested in (Nikolova & Steidl, 2014):

$$\alpha_1 = \alpha_2 = 0.05 \qquad \beta = 0.1$$

Although the variational method is significantly more complicated conceptually and mathematically then the local means method, it was shown to be significantly faster than the local means method (Nikolova & Steidl, 2014). Both methods preserve the ordering of the pixels, but have differences in how they order pixels with the same value.

In either case the strict ordering of all pixels is used to map each pixel to a bin in the target histogram. If the first bin in the target histogram has a value of 0.01 (1%) then the first 1% of pixels in the strict ordering are given the value of the first histogram bin and this is repeated for all histogram bins.

One downside of the exact histogram equalization method is that it requires all of data to be processed at once. This contrasts with the traditional histogram equalization which only requires the source histogram (which can be acquired with a linear pass through all pixels and does not even need all image data in memory at once since multiple histograms can be averaged together) and the target histogram to calculate the necessary linear transformation. Once the transform is calculated it can be applied to any image without needing the original source images, their histograms, or the target histogram. However, the exact histogram equalization methods need to establish a strict ordering for all pixels simultaneously which means that all data needs to be in memory at once causing bottlenecks for larger images. It also makes it impossible to apply the same transformation to a whole series of images, instead the series of images must all be considering simultaneously, greatly increasing the time and memory requirements.

3.3.6.1 Local Means Strict Ordering Improvements

The local means method for creating a strict ordering of pixels is very time and memory intensive, requiring 6 different convolutions that either involve a division or working with floating-point numbers followed by a lexicographical sort of that data. However, a novel method created for this project greatly reduces the resource cost by taking advantage of the increasingly fast 64-bit integer operations built into modern CPUs and the fact that most images use unsigned 8-bit pixel values. All six orders are calculated at once using a single convolution of the image with the following kernel written with hexadecimal values followed by a standard quicksort of the values:

$$\begin{bmatrix} 0x1 & 0x400 & 0x200000 & 0x400 & 0x1 \\ 0x400 & 0x80000000 & 0x20000000000 & 0x80000000 & 0x400 \\ 0x200000 & 0x20000000000 & 0x8000000000000 & 0x20000000000 & 0x200000 \\ 0x400 & 0x80000000 & 0x20000000000 & 0x80000000 & 0x400 \\ 0x1 & 0x400 & 0x200000 & 0x400 & 0x1 \end{bmatrix}$$

This ultimately places each of the order sums is in a consecutive set of bits in the resulting unsigned 64-bit integer output with the higher bits containing the sums closest to the central pixel. Table 3.1 describes which bits are used for each order and the number of bits required for each order. Since the ordering of sums is equivalent to the ordering of means, the integers can be sorted directly with no division or floating-point numbers required.

Even though the variational method was originally shown to be significantly faster than the local means method (Nikolova & Steidl, 2014), this improved algorithm for calculating local means strict ordering is more efficient, as shown in Table 3.2. It is now over 20% faster and uses over 25% less memory

Table 3.1. Bits used by each local sum for strict ordering of pixels for exact histogram equalization. The number of values is the number of pixel values that are summed for a particular order and the number of bits required is the bits required to store the largest value produced by the sum. The bit used are selected to put the lower orders in the mote significant positions. The five most significant bits are left as 0.

Order	Number of Values	Number of Bits Required	Bits Used
1	1	8	5-12
2	4	10	13-22
3	4	10	23-32
4	4	10	33-42
5	8	11	43-53
6	4	10	54-63

Table 3.2. Speed and memory usage of strict ordering methods for exact histogram equalization. Using an unsigned 8-bit integer 22000x16000 pixel grayscale image (352 MP). The time to calculate the strict ordering is only the time required to calculate and sort the pixel values. The total time is the amount of time to read, perform the exact histogram equalization including calculation of the strict ordering, and save the new image. The max RSS is the largest amount of memory used during this process. With no histogram equalization the time and memory is that required for reading and saving the image only. Results were calculated as the average of 10 runs using the `imstack` tool on the hardware described for timing in 5.2.

Strict Ordering Method	Time to Calculate Strict Ordering (mins)	Total Time (mins)	Max RSS (GiB)
None	N/A	0.64	1.05
Local Means (Original)	9.63	10.97	24.32
Variational	3.13	3.64	13.83
Local Means (Improved)	**2.56**	**3.04**	**11.21**

than the published variational method for the same amount of separation of pixels using their strict ordering.

3.3.7 Segmentation Pipeline Toolbox

The ultimate goal for our use of automated segmentation is to not need user intervention to go from microscope to segmented results ready to be quantified. Besides automatic segmentation program there is pre-processing, post-processing, and image format conversions. To do this, we need a set of tools that can be built into a complete pipeline. However, most image processing tools available only work with single images while we care about volumes of images assembled from individual slices of image data. This dichotomy resulted in issues when trying to automate an entire pipeline to go from the microscope to quantified data. Additionally, several of the image formats used by microscopes and segmentation tools are not formats commonly supported by general image processing programs, such as MRC, MATLAB, DICOM, and MetaImage. For this reason, the pysegtools library was developed for this project. This library is written in Python and is designed to deal with stacks and volumes of images and incorporates many pre- and post-processing tools along with the ability to handle several formats of images, from the common formats like PNG, TIFF, and JPEG, to the more specialized formats like MRC and MetaImage.

Nearly all of the functionality provided by pysegtools is exposed through a flexible command line tool called `imstack`. This allows a pipeline to be built by simply executing this program. The arguments to the `imstack` command build off of each other. Each set of arguments represents a single operation to perform and consumes and/or produces one or more image stacks. If an operation consumes an image stack, it takes the most recently produced image stack. Some operations may even consume and/or produce multiple image stacks. For example, the ‘histeq’ operation performs histogram equalization. It consumes one image stack for the images that will be histogram equalized and may take an additional image stack that it uses as a mask which restricts the regions of the image where the equalization will take place. It produces a single image stack which has the images histogram-equalized.

The system is highly flexible, and each image stack can be used for as many operations as needed without having to reload the data. Internally, each operation is only performed on slices or stacks as needed to conserve memory usage, making an overall highly efficient system even when working with large image volumes as are common with SBFSEM datasets. All of the operations and file formats are handled with a plugin system and thus the library is highly extensible.

Since pysegtools is mainly targeted at working with image stacks, it has custom file format handlers for working with image formats that inherently store image stacks, including reading and writing to/from MRC, MATLAB (MAT), and MetaImage (MHA/MHD). Additionally, it is able to read (but not write) from any image stack formats supported by Pillow, including SPIDER, IM, TIFF, and GIF (each frame of animation is considered a separate image in the stack). Image stacks and volumes can be stored as a collection of separate 2D image slices as well, so it supports this as well, with each slice in the stack being either a 2D MetaImage or MATLAB image, or one of the standard image formats supported by the Pillow library, including TIFF, PNG, and JPEG (Clark & et al, 2017).

While some of the operations in the library have yet to be added to work with the 3D data, most of them do use 3D data when appropriate. Additionally, the operations have been written as generally as possible and are designed to work regardless if the pixels are stored as bytes or floating-point values, represent complex data, color, or grayscale information. Obviously, some operations do not make sense for certain types, but where possible all types are supported. When types are not supported, there are several operations for coercing data types as needed.

Some of the more complex operations supported by pysegtools are histogram equalization (approximate, exact using local means, and exact using the variational approach), automatic binary or multi-level thresholding using Otsu's method, connected-components labeling and re-labeling, Canny edge detection, and anisotropic diffusion filtering.

While image processing pipelines will mainly use the command line tool `imstack`, pysegtools also has a rich application program interface (API), allowing other programs to directly hook into the file reading or image processing components used by the command line tool. In fact, the PyCHM command line tools rely on the pysegtools reading and writing of image stacks to be able to open and save a wide variety of image formats to be able to fit into any pipeline itself. This means that if a plugin is written to allow `imstack` to read a proprietary file format for a particular microscope, PyCHM will also be able to read those files as well.

3.4 GLIA: Automatic Segmentation of Neuropil

While CHM-LDNN can classify each pixel as a particular type of object such as a lysosome or nuclei it cannot inherently identify the different objects within a particular class, for example it does not say that this set of pixels is a different lysosome than another set of pixels. For some objects and with high enough resolution a simple connected-components labeling evaluation can be performed which identifies separate objects by separation with pixels that are labeled as not part of the classification (Berengolts & Lindenbaum, 2000). However, this is not always a viable solution for objects that may be touching, separated by less than the resolution of the pixels, or may be surrounded by ambiguous pixels. For these more advanced methods are required. Of particular interest is segmenting the cells in a dataset so that we know which are separate cells. For this purpose, the Hierarchical Neuronal Segmentation (HNS) or Graph Learning Library for Image Analysis (GLIA) created by Liu *et al* (2012; 2014; 2016) at the University of Utah's Scientific Computing and Imaging Institute (SCI). HNS and GLIA are two names for the same concept, with GLIA being the more modern name.

GLIA is designed to operate either as automatic segmentation or as semi-automatic segmentation with input from a user as in the Interactive Merging program (Jones, Liu, Ellisman, & Tasdizen, 2013; Liu, Jones, Seyedhosseini, & Tasdizen, 2014; Liu, Zhang, Javanmardi, Ramesh, & Tasdizen, 2016). Both of these start with the same several steps, but differ in the last few steps. When performing automatic segmentation, as it was used in this project, GLIA is composed of seven major steps:

1. **Pixel-wise membrane detection:** the first step is using another machine-learning system that calculates the probability of each pixel being a cellular membrane. In this project, PyCHM was used to generate these probability maps.
2. **Initial segmentation:** once a probability map of the membranes is obtained an initial segmentation is performed. The goal is to oversegment the data, resulting in every neuron containing several regions but no region that contains more than one neuron. While many are available, the morphological watershed algorithm (Beucher & Lantuéjoul, 1979; Beare & Lehmann, 2006) was chosen in the original paper and due to high computational efficiency and good adherence to image boundaries. The initial segmentation is not generated directly from the probability maps but instead from the Gaussian-blurred probability maps to reduce the number of regions found. Besides the amount of blurring performed, a threshold, called the water-level, is chosen to determine how much of an oversegmenation is performed, causing all boundaries discovered below that level to be ignored.
3. **Pre-merge regions**: the initial segmentation may produce regions which are too small for the extraction of meaningful region-based features in step 5 below. To address this, small regions are merged with one of their neighbors before any further analysis is done. Two criteria are used to decide if a region should be merged with a neighbor at this point: if a region’s area is below some threshold or if a region’s area is below a higher threshold but its average membrane probability is above another threshold. In either case the region is merged with the neighbor that yields the

largest “merging saliency”. The merging saliency is calculated as one minus the median value of the membrane probabilities of pixels along the border between the two regions, producing a higher value when the boundary is mostly unlikely to be membrane pixels and lower when they are mostly likely to be membrane pixels.

4. **Merge tree construction:** a binary tree is created that describes the order in which to merge all of the initial regions with one of their neighbors until there is only one region that describes the entire image. This does not actually commit to merging regions as is done in step 3 but instead creates a hierarchical structure describing how they should be merged, which is used with additional information calculated in later steps to generate the final set of actual merges. The initial regions, generated by steps 2 and 3 above, are the leaf nodes in the tree and the root of the tree is the region describing the entire image. When two nodes should be merged, they are made to be siblings in the tree and connected through an internal node that represent the merged region. The tree is built from the leaves upward by taking the two neighboring regions that have the largest merging saliency among all pairs of neighboring regions.
5. **Feature extraction:** several features are extracted for each region, including all of the initial and potentially merged regions. These features include information about the region geometry such as area, perimeter, and compactness, information about the boundaries that will be eliminated in a merge including the length and curvature, along with summary pixel information along the boundary and within the region from both the original EM images and the probability maps such as mean value. Many possible parameters are available in GLIA to determine which features are utilized and how they are normalized, but only the default set of features and normalizations were used in the project. These features are in general more informative than those used by CHM and other pixel-based classifiers since they are extracted over regions instead of individual pixels.

6. **Classification of boundaries:** a machine learning algorithm is fed the features extracted about the boundaries and regions to predict if two regions meeting at a boundary should be merged or not. The researchers used artificial neural networks (Liu, Jurrus, Seyedhosseini, & Tasdizen, 2012) and random forests (Liu, Jones, Seyedhosseini, & Tasdizen, 2014) when HNS was first published, but in the latest iterations, now called GLIA, they use the same LDNN classifier used by CHM. For this project, LDNN was used as well. These are supervised algorithms and thus must be trained with a set of known samples. Ultimately these only give the probability that any two regions should be merged.
7. **Greedy segmentation:** the final step is to produce the final segmentation from the merge probabilities using a greedy algorithm by assigning "potentials" to each node based on its probability of merging with its sibling and the probability of its children merging with each other.

GLIA is set up as several smaller programs, at least one for each of the steps listed above, that each must be run in a specific order and usually once for each slice of data in the image stack. The overall set of programs is shown in Figure 3.5. This ends up requiring the coordination of thousands of program executions, some of which can be run in parallel and others which have dependencies. Additionally, since the algorithm was being developed and modified by Ting Liu concurrently with this project, as advancements were made the set of programs would change. For example, originally when HNS was released it only worked with individual image slices and required additional steps to link the generated labels across the individual slices (Liu, Jones, Seyedhosseini, & Tasdizen, 2014). Later, when it became GLIA, all of the programs directly supported working on stacks of images, and thus generated consistent labeling across slices inherently.

All of this made it impossible to coordinate manually so a generalized coordination system was created. This library that coordinates tasks was written in Python and included in the pysegtools library. It organizes a list of programs that need to be executed given what data each one requires and generates, automatically discovering which programs must run before other programs and which ones can be run simultaneously. It then is able to execute these tasks efficiently knowing what resources are available.

One change made to the published GLIA was the update of the training process to use the LDNN written for PyCHM instead of the one written in MATLAB for CHM. This increases the efficiency of the program without any significant change in the results. Additional utility programs were created for the pre-processing and post-processing of the data, allowing CHM results to be fed directly into the GLIA process and allowing the generated label images of GLIA to be converted into contours around cells that are usable as IMOD models and can be meshed into 3D models.

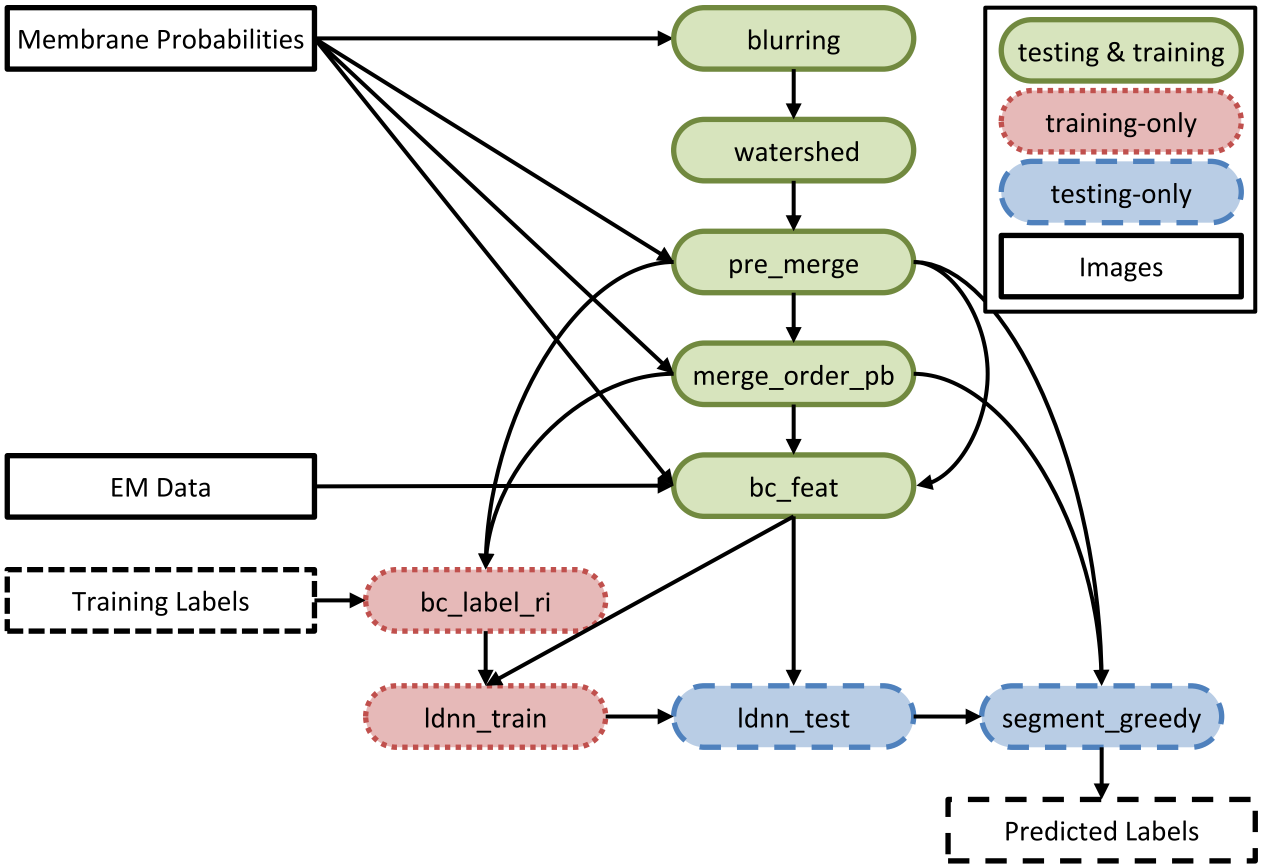

Figure 3.5. GLIA programs and interdependencies. Rectangles are input or output images for the entire GLIA process. Solid green outlines are steps used for both training and testing phases. Red dotted outlines are steps used for training phase only. Blue dashed outlines are steps used for testing phase only. When running in 2D mode, all programs except ldnn_train and ldnn_test need to be run on each slice individually.

3.5 Summary

Several different segmentation algorithms are used in concert to be able to take the raw data from an electron microscope and get segmented cells and organelles. The process to segment organelles is diagrammed in Figure 3.6. From the microscope, a subset of the data must be selected as the training data. This volume is manually segmented in a tool such as IMOD with the assistance of the Livewire tool to accelerate the process. The manually segmented data must be converted from contours around objects to a label image where pixels that are within objects of interest are white. This can be done with the `imodmop` command line program.

The training label images along with the preprocessed training image volume are then given to PyCHM which generates its conceptual model of what it learns distinguishes the objects from the background. Once the model is created, PyCHM can be used to test other volumes of preprocessed data from the same source. The testing process outputs a probability map image which assigns every pixel the probability it is the object that it learned. The probabilities are then thresholded to provide a binary yes or no answer.

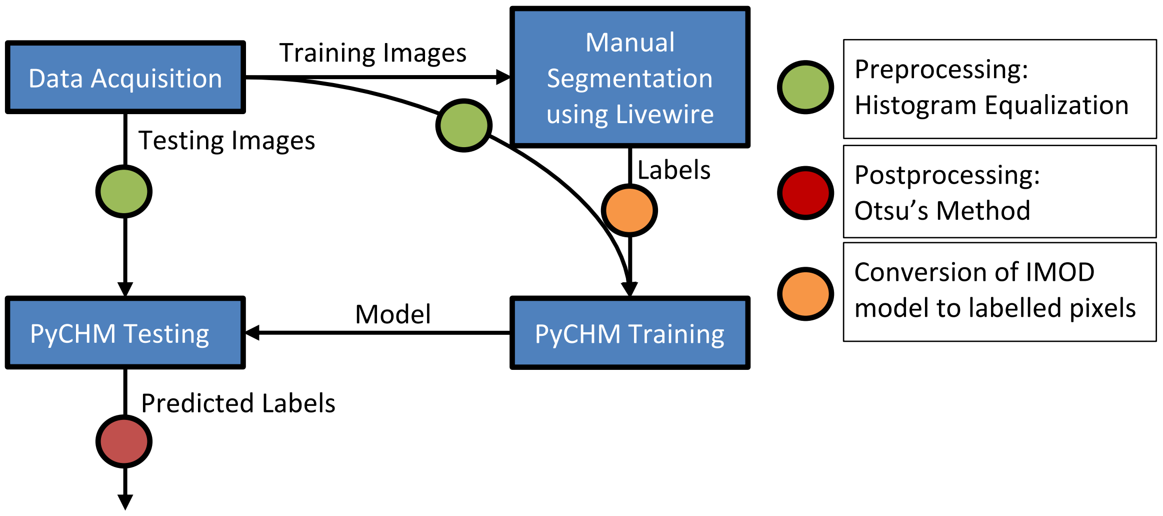
Data Acquisition
Training Images
Manual Segmentation using Livewire
Testing Images
Labels
PyCHM Testing
Model
PyCHM Training
Predicted Labels
Preprocessing: Histogram Equalization
Postprocessing: Otsu's Method
Conversion of IMOD model to labelled pixels

Figure 3.6. Process to automatic segment of organelles using PyCHM. For segmenting a single organelle from a single dataset. The preprocessing and postprocessing steps are completed using `imstack`. Conversion of IMOD model to labels is completed with `imodmop`. Manual segmentation is done within the graphical user interface of IMOD.

To segment entire cells, first cell membranes must be segmented with PyCHM as described above except that the final step to threshold the probabilities to obtain binary results is not performed. Instead, the probabilities and labels are inputted into GLIA for segmentation of cells. This process is diagrammed in Figure 3.7.

The algorithms described in this section have all been explicitly modified or created for the project. Table 3.3 provides a summary of the algorithms and programs along with the developments introduced for this project and where the code for each can be obtained.

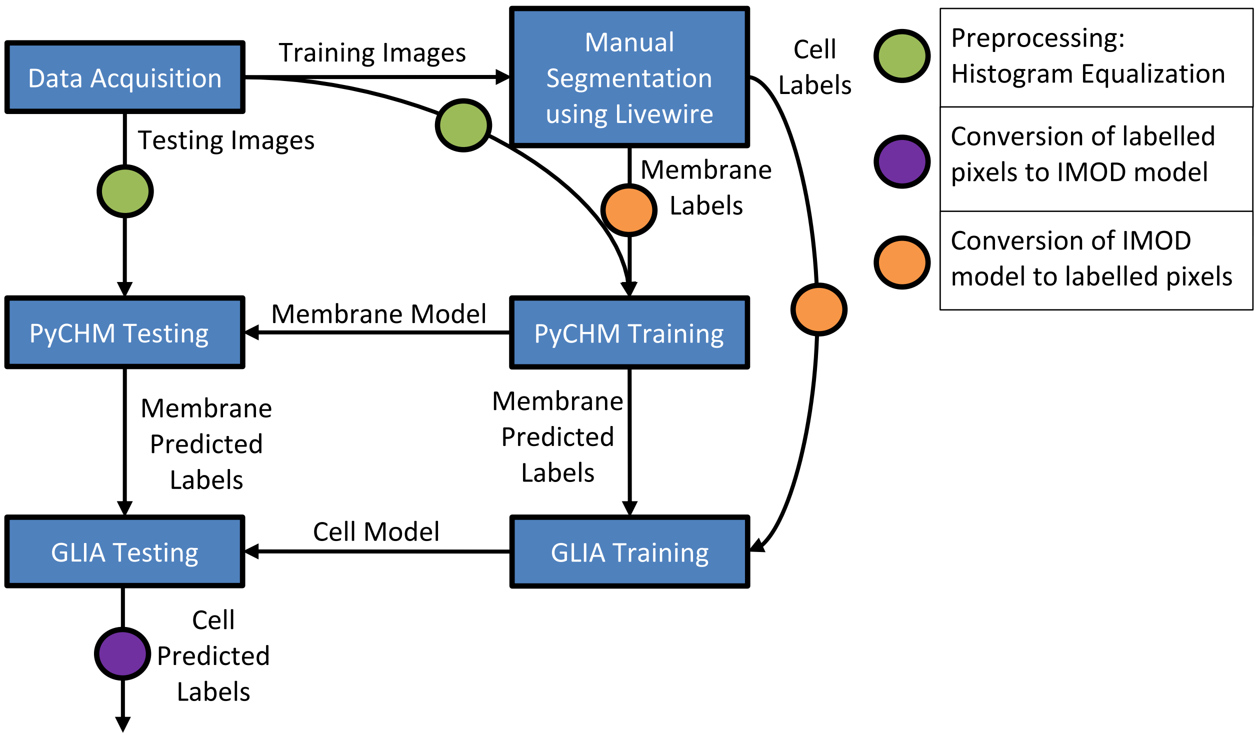
Data Acquisition
Training Images
Manual Segmentation using Livewire
Cell Labels
Testing Images
Membrane Labels
PyCHM Testing
Membrane Model
PyCHM Training
Membrane Predicted Labels
Membrane Predicted Labels
GLIA Testing
Cell Model
GLIA Training
Cell Predicted Labels
Preprocessing: Histogram Equalization
Conversion of labelled pixels to IMOD model
Conversion of IMOD model to labelled pixels

Figure 3.7. Process to automatically segment entire neurons using PyCHM and GLIA. PyCHM is used to segment the cell membranes and GLIA uses that information to segment the individual cells from a single dataset. The preprocessing step is completed using `imstack`. Conversion of IMOD models to labels is completed with `imodmop`. Conversion of labelled pixels from GLIA back to an IMOD model is completed with a custom Python tool created for GLIA and the IMOD program `point2model`. Manual segmentation is done within the graphical user interface of IMOD.

Table 3.3. Segmentation programs developed for this project. The developments made to each of the segmentation programs and the basic image processing toolkit for this project along with where the source code can be obtained.

Program	Developments	Availability
Livewire	Completely new implementation written in C++ from scratch with several innovations to be able to handle massive EM images efficiently and accurately; integrated it into IMOD for general usability	IMOD version: bio3d.colorado.edu/imod/source.html Standalone version: github.com/slash-segmentation/livewire
PyCHM	Completely new implementation of CHM and LDNN written in Python from scratch that improves accuracy, speed, and memory usage from the original CHM and LDNN; added additional features including normalization of features, the Frangi filter, and masking	github.com/slash-segmentation/PyCHM
GLIA	Updated to use re-written LDNN, added tools written in Python to organize the execution of the myriad of tasks, convert label images to models, and calculate paired-pixel statistics	github.com/coderforlife/glia
pysegtools and `imstack`	Written in Python from scratch to handle large image volumes, read and write specialized image formats, and perform necessary preprocessing and postprocessing including automatic thresholding using Otsu's method and exact histogram equalization; created a new exact histogram equalization approach that is more efficient than previous approaches	github.com/slash-segmentation/segtools

4 GeoEM: Geometry Analysis of EM Data

GeoEM is a program developed for this project for analyzing the <u>geo</u>metry of <u>EM</u> data. The goal of the program is to robustly measure various metrics about cells and organelles in the segmented EM data. For example, previous studies looking at the taper of branches leaving a branch point used hand-drawn estimations from a single 2D image (Ferrante, Migliore, & Ascoli, 2013). This method has issues with reproducibility when different people select different starting points to measure the taper for each branch. Instead, if we can analyze the 3D mesh of the cell itself then a robust and reproducible metric for taper can be calculated.

Besides the measurement of properties of the branching structure of the dendrites, the mesh can be lined up with segmentation data of organelles such as mitochondria and ER. Then the density of these organelles can be analyzed based on their location within the mesh – are the organelles near a branch point or are they clustered in the middle of a branch? Are they near the axis of the dendrite or are the near the surface?

GeoEM makes extensive use of the Computational Geometry Analysis Library (CGAL) which is an open source C++ project that provides access to efficient and reliable geometric algorithms (The CGAL Project, 2018). The library is divided into various parts. Besides the core functionality the main parts used for this project are: the 3D Polyhedral Surface and Halfedge Data Structures (Kettner, 2018; Kettner, 2018), Surface Mesh Generation (Rineau & Yvinec, 2018), Volume Mesh Generation (Alliez, et al., 2018), Polygon Mesh Processing (Loriot, Tournois, & Yaz, 2018), Surface Subdivision Methods (Shiue, 2018), and Surface Mesh Simplification (Cacciola F. , 2018), Surface Mesh Skeletonization (Gao, Loriot, & Tagliasacchi, 2018). CGAL heavily utilizes the open source Boost Graph Library (BGL) (Siek, Lee, & Lumsdaine, 2001; Cacciola, Moeller, & Wein, 2018) along with other portions of the Boost C++ Libraries. It also uses QGLViewer (Debunne, 2017) with OpenGL (Khronos Group, 2017) and Qt (The Qt Company, 2017) to display 3D models and libpng to read image data from PNG files (Schalnat, Dilger, Bowler, & Randers-Pehrson, 2017).

The geoEM program reads a 3D mesh of segmented cells from EM data along with probability maps for an organelle such as mitochondria or endoplasmic reticulum that overlap with the segmented cell. The cell is assumed to be a branching structure like a neuron. Each branch is divided into smaller "slices" with each slice being an approximate cylinder along the branch, with either one or two open ends. The branch points themselves are also extracted as slices, however they are not limited to being cylindrical and can have multiple open ends (typically 3, one towards the soma and two towards the distal ends of the dendrites). The slices also store information about their connectivity with the other slices. This connectivity information is used to calculate the direction to travel towards the soma, the distance to the soma, and the next branch point that will be reached traveling towards the soma. Distances are measured along the "skeleton" of the mesh and start from the middle of the slice.

Additional metrics can be calculated at this point, including surface area, volume, and surface area to volume ratio of the slices. For surface area and volume, these metrics are not useful as-is since each slice may be approximated by a cylinder with a different height (although it tries to keep them similar, it cannot do so perfectly). Instead, a normalized volume and surface area is more useful, where each of the metrics is divided by the height of the cylinder (i.e. the distance along the skeleton from the top to bottom of the slice).

These metrics calculated from the mesh provide insight into several pieces of important information about branches within a neuron which includes the calculation of the geometric ratio of a branch point. Moreover, with the level of detail provided in EM data, the location of organelles relative to the branch points can be calculated. This is accomplished by determining which pixels in the image data are within each slice and summing the thresholded probability values for those pixels.

Many of these steps unitize functions included in CGAL but other functions are specific solutions for this project. The detailed steps of this process are:

1. Read 3D mesh from Wavefront OBJ or OFF (Object File Format) file. Both of these formats are commonly used for 3D models. The program command-line `imod2obj` included with IMOD converts an IMOD 3D model to OBJ.
2. The skeleton of the 3D mesh is constructed using the mean curvature algorithm (Tagliasacchi, Alhashim, Olson, & Zhang, 2012) which works by iteratively contracting the mesh to a series of line segments that lie in the middle of the mesh that are connected at vertices to other segments resulting in an undirected acyclic graph. Vertices along the skeleton of the branch will be of degree 2 while skeletal vertices at branch points will be of degree 3 or higher. Vertices at the ends will have a degree of 1. This uses the `CGAL::Mean_curvature_flow_skeletonization` class. The only value not set to the default is the quality/speed tradeoff (w_H in the paper), increasing it from the default of 0.1 to 0.5. A representation of a skeleton is shown in Figure 4.1.
3. The mesh is then sliced. The only parameter for this step is the number of skeleton vertices to group together for each slice, either increasing the resolution of the slices if a small value is chosen

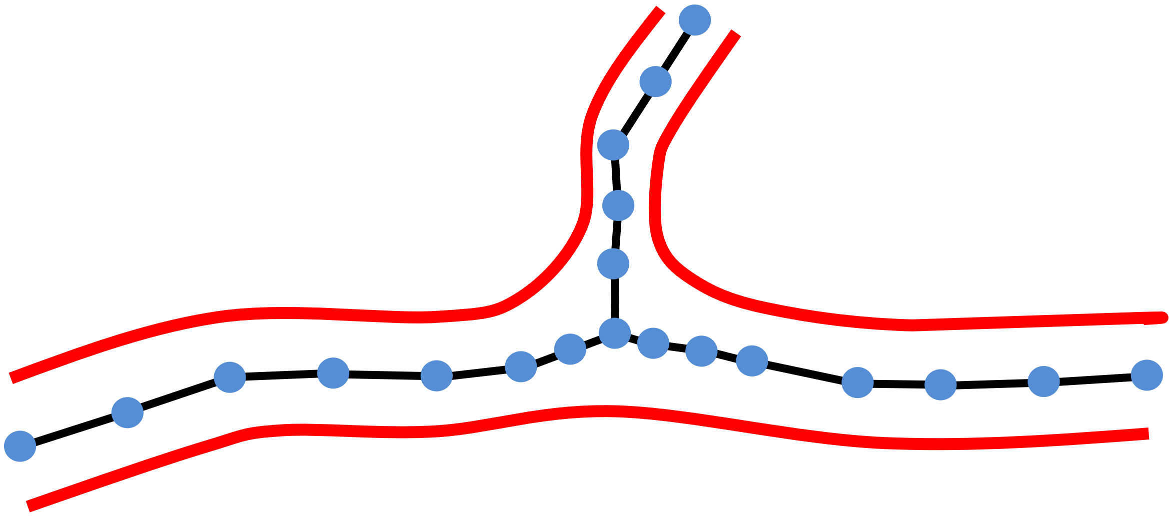

Figure 4.1. 2D representation of a mesh with the skeleton. The mesh is the outer red line. skeleton is the black line segments through the middle. The skeleton vertices are the blue circles. This example has a single branch point in the middle.

or increasing the consistency of the slices if a larger value is chosen. Since this step is the most involved in terms of implementation for the project, it is detailed below.

4. The “soma” is then found and distances to each slice are calculated. The soma is defined as the slice with the largest normalized volume that is at the end of a branch and only has one neighbor (i.e. a terminal of the skeleton). This is performed with a simple search over all slices. The distance to each slice, traveling along the skeleton, from the soma is then calculated. This is performed using a depth-first-search starting from the soma. Once the distances are calculated another pass is performed to identify the proximal branch point (or soma if along the trunk before any branching occurs). This is done with a recursive function that continually travels from each slice towards the soma until a branch point or the soma is found, or a slice for which the nearest branch point was already calculated.
5. The probability map image volume for the organelles is loaded. Besides the image types supported by CGAL, all of which are either esoteric or limited in functionality, support for MRC files and stacks of PNG files is added. These are the common formats used in the rest of the segmentation pipeline. MRC files are IMOD’s native format and PNG files providing a wide range of features, including compression, but not able to store more than one slice.
6. Finally, the percent of volume of each slice is calculated. This is done by counting all voxels (volume-pixels) whose center location spatially lies within the slice and whose probability map value is greater than a given threshold. Since the dataset is not isometric (i.e. has unequal lateral and axial resolutions) careful work must be done to align the voxels with the slices, considering the spacing of the voxels in each direction. The number of found voxels is then multiplied by the volume of a single voxel (based on the spacing between voxels) and divided by the volume of the mesh of the slice.

The last step could be done by taking the volume of the intersection of two meshes: the slice and a mesh of the organelles. However, that mesh itself would be an approximation itself and could never be more accurate than the voxel-wise data. Additionally, creating meshes from intricate regions, such as ER, is extremely difficult, especially since the meshes need to be a not self-intersecting manifold. Specifically, most of the mathematical analysis tools in CGAL require the mesh to have consistent orientation of the faces in the mesh, every face must be a non-degenerate triangle, no face can intersect another face except along an edge, no edge can be shared between more than two faces, and every vertex must have a single sequence of connected faces going around it. Some other requirements for the complete analysis are that the mesh must be closed (i.e. contain no holes) and there must only be one connected component. The analysis of more than one neuron the program must be run on each independent neuron.

Even when generating meshes on larger structures such as whole neurons, the meshes still may not meet all of these requirements. IMOD frequently generates meshes with small defects in them, likely due to floating-point rounding errors, that cause some faces to intersect other faces. Also, it makes no attempt to orient all faces consistently. To address this an additional program was created using CGAL to repair meshes and create models that are usable for analysis. The repair process is as follows, using several functions from `CGAL` and `CGAL::Polygon_mesh_processing`:

1. Load all the faces from a mesh, stored either as an OBJ or OFF file, while ignoring any connections between them. This is called a “polygon soup” instead of a polyhedron.
2. If any of the faces are degenerate, then they are completely removed. Any of the faces that are non-triangular are triangulated with the function `triangulate_hole_polyline`.
3. The polygon soup is consistently oriented and converted into a polyhedral mesh with the functions `orient_polygon_soup` and `is_polygon_soup_a_polygon_mesh`.
4. All faces that participate in self-intersections are completely removed from the mesh. They are found with the function `self_intersections` and then erased from the polyhedron. This

operation may change the shape of the mesh. However, as long as the mesh was already fairly reasonable the number of self-intersecting faces has always been seen to be quite low and composed of relatively small faces.

5. If the polyhedron now has holes in it, either from the start or because faces were removed in the prior step, they are filled in using the `triangulate_hole` function if possible.
6. All but the largest connected component of the mesh is removed.
7. At this point the mesh should be a single closed connected component with valid and outwardly oriented faces. However, if the mesh has more major issues it may not be valid. If the faces are not outwardly oriented (checked with `is_outward_oriented`) then they are reversed with `reverse_face_orientations`. Then self-intersections are removed more aggressively by not only removing the intersecting faces but also their direct neighbors. Since new holes may have been created, the holes are filled again.
8. Finally, the mesh is saved to a new file, either as an OBJ or OFF file.

This process is sufficient to fix meshes of cells created by IMOD but is unable to fix highly convoluted models like those of ER.

To improve the results, one additional step is required so that the facets are all of similar shaped and sized. IMOD creates meshes from contours that lie on separate slices and therefore the meshes have highly skewed distributions of triangular size and shape of the faces. An additional program was created to refine the meshes after they were repaired. The program can refine the mesh in three different ways:

1. Isotropic remeshing to make each of the faces more regular using the function `CGAL::Polygon_mesh_processing::isotropic_remeshing` which implements the algorithm described by Botsch *et al* (2004; 2010). This maintains a good approximation of the original mesh while adjusting the edges in the polyhedron so each edge ends up with approximately the same length. This makes every face approximately an equilateral triangle. If

the target edge length is longer than the current average edge length this will cause a simplification of the mesh to occur, reducing the number of faces significantly to achieve the target edge length.

2. Performing Loop subdivisions to increase the number of faces in the mesh while keeping the shape. This uses the function `CGAL::Subdivision_method_3::Loop_subdivision` function implementing the subdivision method by Charles Loop (1987).
3. Edge collapsing to simplify the mesh and reduce the number of faces while keeping the shape. This uses the function `CGAL::Surface_mesh_simplification::edge_collapse` with a policy to stop either when the number of edges reaches a certain amount or when the proportion of current edges to original number of edges reaches some threshold.

Typically, isotropic remeshing is the most important refinement process as it adjusts the faces to become equilateral triangles while also being able to either simplify or increase the detail of the mesh. However, the other methods tend to retain the shape more consistently, so isotropic remeshing can be used to equalize the shape and size of the worst faces and then use the other methods to adjust the quality of the mesh.

The code for geoEM, including the utilities for repairing and refining polyhedral meshes, is available at github.com/slash-segmentation/geoEM.

4.1 Slicing

Calculation of any of these metrics, the mesh must be sliced into pieces. Many methods of slicing the mesh were explored before settling on the one to be used. The other methods had significant issues with closing each of the slices (necessary to calculate volume and determine if a point lies within the slice), not representing the original mesh accurately, or creation of slices that were not approximate cylinders.

The initial step is grouping the skeleton vertices. The branch points in the skeleton are placed into groups based on how large the mesh is within their region. The branch point skeleton vertex and its immediate neighbors are always included in the group. Additionally, all skeleton vertices connected to the group that are less than the distance of the first neighbor to the mesh away from that neighbor are included as shown in Figure 4.2. This makes sure that branch point slice will include most of the region that is not part of any of the branches going out from the branch point. It will automatically scale to the size of the branch point – small branch points will only include the core group of skeleton vertices while larger branch points can include several additional skeleton vertices. Within a single branch point the number of skeleton vertices added in each direction may be different since each of the neighbors is considered independently. This means that if a small branch is coming off of a smaller trunk, it will automatically adjust for it as well.

After the branch point groups are created, the other skeletal vertices are grouped together with their neighbors. The maximum size of a group is a parameter that controls the number of consecutive

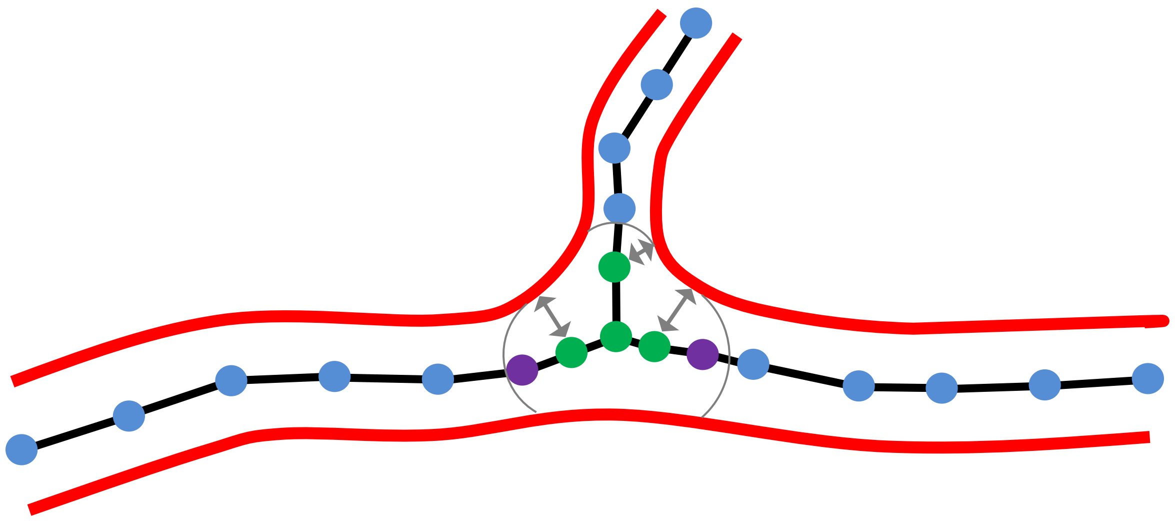

Figure 4.2. Grouping of skeleton vertices for branch points. The core group of skeleton vertices for a branch point that is always included (the branch point skeleton vertex and its immediate neighbors) are shown in green. Their minimum distance to the mesh (red) is shown with gray arrows. All neighboring vertices of the core group that lie within this distance (purple) are included in the group of skeleton vertices for the branch point. Each direction away from the branch point is considered independently of the other directions. The branch going upwards does not include any additional vertices while the other two branches each include an additional skeleton vertex.

skeletal vertices placed in a group, and thus the amount of detail that is preserved. Making the groups too small leads to issues in creating reasonable cylinders for each slice while making groups too large would not preserve the detail that is given by using EM data in the first place. When a branch between two branch points cannot be broken into slices that are each the maximal size then some of the groups will have one less vertex to make the division work. The groups that are made to have one less vertex are the groups that would end up being the largest in terms of number of mesh vertices that are associated with the skeletal vertices. In some edge cases when a branch is very short then none of the groups may have the maximal number of skeletal vertices.

The groups created so far are groups of skeletal vertices. These groups are then made into slices which are defined by a set of planes intersecting the mesh. A plane is defined by each of the terminals of the group as the plane that is orthogonal to the skeleton through the midpoint of the skeleton edge between a skeleton vertex in this slice and the neighboring slice. Only parts of the mesh loosely associated with the skeleton vertices in the group are considered to avoid taking portions of the mesh that are part of other branches.

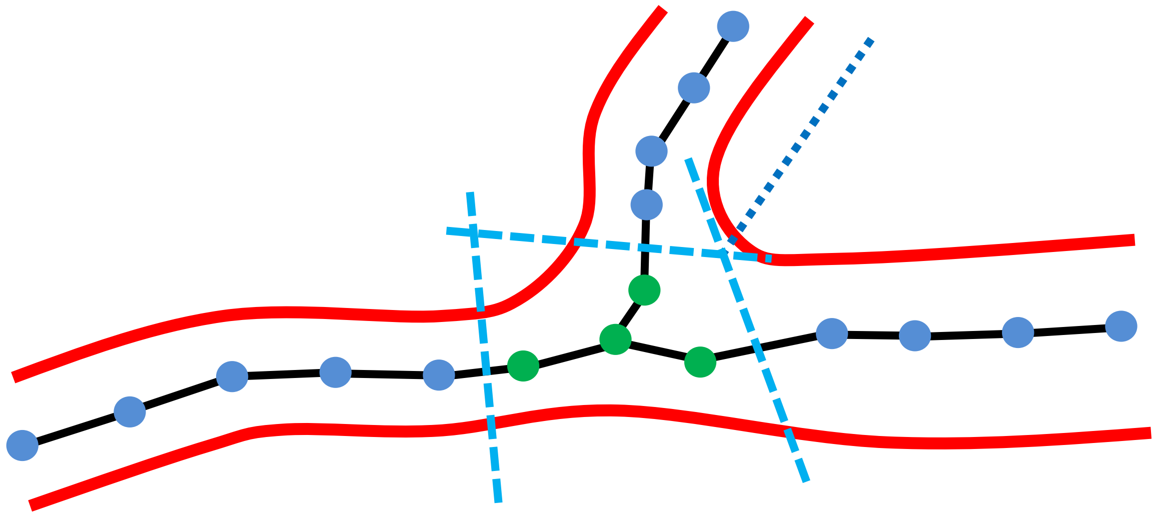

Figure 4.3. Additional planes used around branch points. The planes that define the branch point slice are shown with dashed blue lines. Using these planes, the slices to the right of the branch point slice will "leak" into the slice at the top of the branch point and end up including the entire upper branch since it only uses the left-most plane to separate it from the branch point slice. In this situation, an additional plane is added, shown here as the dotted blue line. This new plane bisects two of the other planes, separating the top and left slices from each other.

One additional consideration must be taken into account. Near branch points, two planes may not be sufficient to isolate the relevant parts of the mesh from the neighboring slices. This occurs if the branch point group is not quite large enough to encompass the entire branch point region. Additional planes are also given by nearby branch points to accommodate this. These planes bisect any two of the planes that define the branch point slice. This is shown in Figure 4.3. While this example is a bit contrived to work in 2D, in 3D these situations are fairly common. An example of these in a real mesh is shown in Figure 4.4.

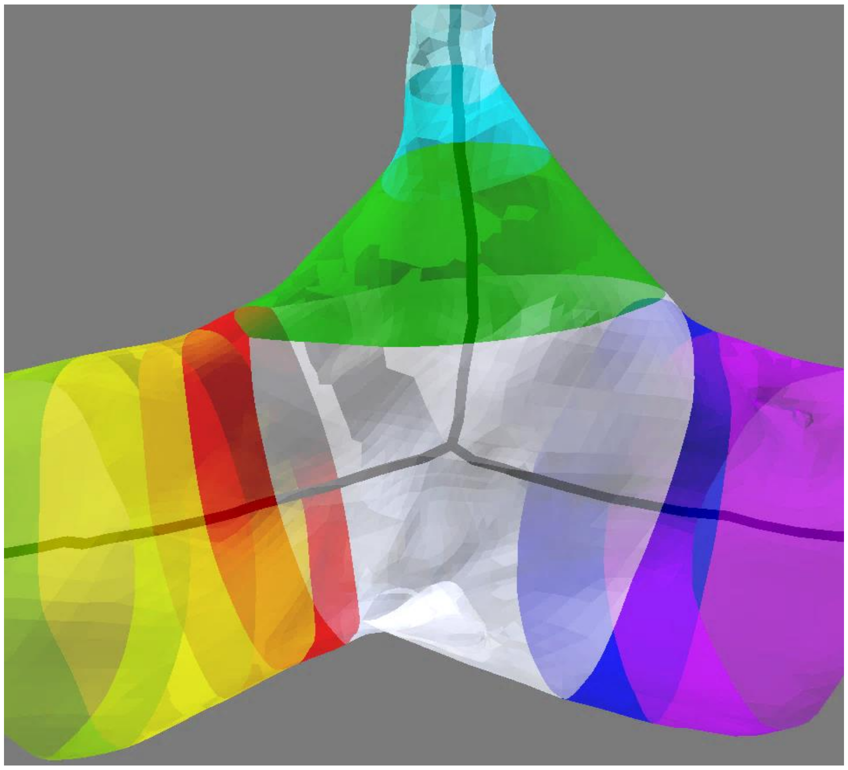

Figure 4.4. Example of necessity to use additional planes around branch points. The green slice (above the branch point) and the red slice (to the left of the branch point) meet at a plane that bisects two of the planes that define the branch point.

5 Methods

In this section we will discuss the datasets and methods used to verify the quality of results from the new PyCHM and with GLIA along with the use of these tools on larger datasets to automatically segment novel regions within the datasets.

5.1 Datasets

The datasets used for verifying the quality of results are listed in Table 5.1 and Table 5.2 along with their resolution, number and size of images, and number of samples of each label. Additionally, the *neuropil* dataset is used to automatically segment large cells for use with geoEM.

Table 5.1. Dataset properties. The training images are those used for training the CHM model (or GLIA model for the neuropil dataset). The testing images are used to verify the quality of the model. The neuropil training and testing images are combined to form a larger training dataset when attempting to automatically segment unknown regions of data.

	Resolution (nm/px)		Training Images		Testing Images	
Dataset	Lateral	Axial	Count	Size	Count	Size
Lysosomes	7.8	30	50	500x500	40	1000x1000
Mitochondria	7.8	30	50	500x500	40	1000x1000
Nuclei	7.8	30	50	500x500	40	1000x1000
Nucleoli	7.8	30	50	500x500	40	1000x1000
Neuropil	5.2	35	35	601x601	21	601x601

Table 5.2. Dataset label counts. For each label of each dataset the number and percent of labeled pixels is given for both the training and ground truth (testing) image sets. These labeled pixels were generated using manual/assisted segmentation and were used in training the models or verifying their accuracy with testing.

		Training Labels		Ground Truth Labels	
Dataset	Label	Pixels	Percent	Pixels	Percent
Lysosomes	lysosomes	116282	1%	96533	0.2%
Mitochondria	mitochondria	701232	6%	635199	2%
Nuclei	nuclei	5226805	42%	12663477	32%
Nucleoli	nucleoli	1363553	11%	1107531	3%
Neuropil	cell membranes	2934958	23%	1609461	21%
	mitochondria	755504	6%	435452	6%

The first four datasets (***lysosomes***, ***mitochondria***, ***nuclei***, and ***nucleoli***) were from (Perez, et al., 2014) and included manually traced labels for each of the organelles. They were already divided into a training set and a testing/ground-truth set, used to verify the accuracy of the model learned by CHM. These datasets do not include entire cells, and therefore they were only used for verification of the features of the new PyCHM compared to the original CHM.

The final dataset, ***neuropil***, was specifically acquired for this project for the segmentation of whole cells within a 3D volume of neuropil. This allows for testing the entire process of PyCHM, GLIA, and geoEM. The data was collected on a Zeiss Merlin SBFSEM microscope using a block of neuropil from the CA1 region of a rat hippocampus. It was positioned so that the somas are at one end of the field of view so the volume can be oriented. It was stained using the same procedure used for the other datasets (Perez, et al., 2014). It was acquired using a 1.5 KeV acceleration voltage. The resolution is 5.2 nm/px laterally and 35 nm/px axially with dimensions $22000 \times 16000 \times 871$ px giving a total tissue volume of 2.9×10^5 μm^3 $(114.4 \times 83.2 \times 30.5$ μm).

A $601 \times 601 \times 56$ px sub-region of this dataset was selected and all mitochondria and cells were traced using manual and assisted segmentation tools within IMOD. To check the accuracy of PyCHM and GLIA, the first 35 slices were used for training and the last 21 slices for verifying the accuracy. However, when automatically segmenting larger regions of the volume, the entire set of 56 slices was used for training so that CHM and GLIA had a larger volume of information for learning. Other, larger, subsets of the data were used to run tested models and to obtain automatically segmented cells.

All datasets were preprocessed using exact histogram equalization using the variational method with five iterations utilizing the `imstack` program. All image stacks were stored either as single MRC files or as folders of PNG files. In both cases, unsigned 8-bit integers were used to store the pixel values.

5.2 Computational Hardware and Software

The programs were run on a dedicated custom workstation to measure the timing and speed improvements of CHM and histogram equalization. This workstation ran Fedora 27 Workstation with two Intel Xeon Gold 5122, each with 4 cores (8 threads) at 3.6 GHz, and a total of 256 GiB DDR4-2666 RAM. All file I/O was performed on an NVMe SSD. The memory was setup so that the memory bus of the first CPU had 64 GiB of RAM while the other memory bus had 192 GiB. The processes were pinned to the cores of the CPU with more RAM to minimize the effects of non-uniform memory access (NUMA). They were only allowed to use hyperthreads when running with more than 4 computational threads. PyCHM and `imstack` used the following software: Python v2.7.14, NumPy v1.14.0, SciPy v1.0.0, Cython v0.27.3, gcc v7.2.1, MKL 2018 update 1, pyFFTW v0.10.4 with FFTW v3.3.7, Pillow v5.0.0, h5py vv2.7.1, and psutil v5.4.3. The original CHM used the following software: MATLAB R2017b (v9.3), Image Processing Toolbox v10.1, Parallel Computing Toolbox v6.11, Java SE 8 update 121, and gcc v4.9.2. Whenever possible, code was compiled with the highest level of optimization (e.g. using the `-O3` argument for `gcc`).

The Gabor filter in PyCHM uses the FFTW library which requires significant computational time and memory to develop its "wisdom" for any particular image size, type, and number of computational threads and then caches these results for future use. This will greatly increase the time required for the first training run on a machine relative to the future runs. A script was developed to equalize this that ran FFTW prior to the running of PyCHM given a particular set of image sizes, contextual levels, and number of threads so the wisdom calculation could occur outside of any of the timed runs. This script is built into the Python code for the Gabor filter.

The CPU user time and system time (times the CPU spent processing), the wall time (actual time elapsed), and maximum resident set size (RSS, physical memory usage) for most runs was acquired with the `/usr/bin/time -v` command or equivalently the `wait4()` POSIX system call. However, for PyCHM testing, each task (including the first or only task) is run in a separate process and the program and system call above only acquire the memory usage for the parent process. In these situations, a custom Python script was used that polled the system periodically for the sum of memory usage across the parent and all child processes. The average of the times and the maximum of the memory usage was recorded for each run.

When not measuring time and memory usage accurately, a variety of different hardware was used with potentially different versions of the software listed above. The four computational resources used beyond the custom workstation were:

- The shared, large-shared, and compute nodes of the Extreme Science and Engineering Discovery Environment (XSEDE) Comet cluster at the San Diego Supercomputer Center (SDSC) through allocation ddp140 (Towns, et al., 2014). This is supported by National Science Foundation grant number ACI-1548562.
- The National Biomedical Computation Resource (NBCR) Rocce cluster at the University of California San Diego (UCSD). This is supported by grants from the National Center for Research Resources (5P41RR008605-19) and the National Institute of General Medical Sciences (P41 GM103426) from the National Institutes of Health (NIH).
- National Center for Microscopy and Imaging Research (NCMIR) compute environment at UCSD.
- Amazon Elastic Compute Cloud (EC2) r4.8xlarge nodes. This allocation was supported through NIH Commons Credit Pilot Award (CCREQ-2017-03-00017).

5.3 CHM

Only the *lysosomes* dataset was used to test the memory and speed improvements of CHM. Both versions (MATLAB and Python) were run on the dedicated hardware. The average time and maximum resident set size (RSS) across three runs were recorded. CHM testing operates on single slices and therefore each run consists of 21 time and RSS values.

All other runs were computed with whatever computational resources were available at the time. These were usually shared hardware and therefore even though their run times were recorded, there is likely to be large variations which may not be directly comparable with the other timing values.

CHM training was run with the following settings except where otherwise noted: use 2 stages and 4 levels for the contextual information processing, do not include the Frangi filter or Gabor filters, do not use subsampling, and utilize median-mad feature normalization (only available in PyCHM, original CHM never performs normalization of the features). Testing with the original CHM used a border size of 50 pixels around each tile and the tile size was set to be 100 pixels larger than the image itself so that only one tile was computed which included the entire image to avoid any seam effects in the middle of the images except when specifically testing the memory and speed.

Various features that were added to PyCHM were examined. These included the use of the Frangi filter, use of the Gabor filter, the different feature normalization methods, and whether to use subsampling. If subsampling was used, at most 6,000,000 samples were allowed. These settings only apply to PyCHM and not the original CHM except for changing subsampling. The datasets were put through several (3 to 10) runs of each combination of options to find the highest average G-mean value. All future runs for that dataset used the best found set of options.

Testing of the masking feature was performed by training and testing on cropped versions of the organelle datasets and comparing the results against the full datasets but masking out the cropped regions. The training images were cropped/masked down to the central 360x360 and 400x400 regions. The number of labeled pixels in each set is given in Table 5.3. The testing images were used unmodified. Masking was also explored with the *neuropil* dataset by training and testing first with mitochondria labels and then training and testing with cell membrane labels with the mitochondria masked out.

Table 5.3. Cropped/Masked dataset label counts. These central regions of the training datasets are used to test the masking feature of PyCHM, allowing for the training process to utilize the surrounding pixels for feature extraction.

		Central 360x360		Central 400x400	
Dataset	**Label**	**Labeled Pixels**	**Labeled Percent**	**Labeled Pixels**	**Labeled Percent**
Lysosomes	lysosomes	108296	2%	111962	1%
Mitochondria	mitochondria	451959	7%	523799	7%
Nuclei	nuclei	2966191	46%	3572335	45%
Nucleoli	nucleoli	1311281	20%	1345788	17%

Unless otherwise specified, results were postprocessed using Otsu's single-threshold as implemented in `imstack`, giving binary labels.

5.4 GLIA

GLIA is needed to automatically segment the cell membrane probability maps into compete cells. Then the geoEM tool can be used to analyze the geometry of the generated models. GLIA uses probability maps with values from 0.0 to 1.0 where a value of 1.0 indicates a high certainty of being a membrane. While CHM supposedly creates probability maps, since it is targeting the values 0.1 and 0.9 as false and true instead of 0.0 and 1.0, they are not really probabilities. The accuracy of GLIA is increased by stretching the output of CHM so that 0.1 is moved to 0.0 and 0.9 is moved to 1.0. All values that are decreased below 0.0 are set to 0.0 and all values increased above 1.0 to 1.0. This transformation can be defined as applying the following function f to every pixel in the image:

$$f(x) = clip\left(\frac{5}{4}x - \frac{1}{8}, 0,1\right)$$

$$clip(x, a, b) = \begin{cases} a & x < a \\ b & x > b \\ x & otherwise \end{cases}$$

Additionally, since GLIA expects a probability map and not a binary image, these maps are not thresholded using Otsu's method.

Table 5.4. GLIA parameters explored. The parameters, as per (Liu, Jones, Seyedhosseini, & Tasdizen, 2014) that were explored in finding automatically segmented cells in the neuropil dataset. The value N/A for $t_{\alpha 2}$ indicates that that no regions were pre-merged based on their average probability.

GLIA Parameter		Values Explored
σ	Std. dev. of Gaussian blurring	0.0 (no blurring), 0.5, 0.75, 1.0, 1.25, 1.5
t_w	Watershed initial local minimum dynamic threshold	0.01, 0.02, 0.03, 0.05, 0.07, 0.075, 0.08, 0.09, 0.1, 0.11, 0.125
$t_{\alpha 1}$	Minimum region area threshold for pre-merging	50, 100, 150
$t_{\alpha 2}$	Maximum region area threshold for pre-merging	N/A, 150, 200, 250, 350

GLIA has numerous parameters that can be modified, many of which greatly affect its results. Some of these parameters were explored to find the set for which the best results were produced. The values that were explored are shown in Table 5.4.

The output from GLIA is an image stack where each pixel is given a value indicating the cell it belongs to. In this form it is not particularly useful. Instead, the boundaries around each numbered region is converted into a set of points representing a contour around the object in each plane. A program to accomplish this efficiently was specifically written for this project. These contours can then be converted into an IMOD model using the command line program `point2model`. Then IMOD can be used to mesh each of the objects using the command line program `imodmesh` with the extra argument `-T` which increases the accuracy of the model but also increases computation time.

5.5 Segmentation Algorithm Evaluation

The quality of results of the segmentation algorithms was determined using three different metrics: pixel accuracy, F-value, and G-mean. All of these are defined using the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) which are computed as follows:

$$TP = \sum_i [L_i \wedge G_i] \qquad FP = \sum_i [L_i \wedge \neg G_i]$$

$$FN = \sum_i [\neg L_i \wedge G_i] \qquad TN = \sum_i [\neg L_i \wedge \neg G_i]$$

where L_i is the label of the i^{th} pixel, G_i is the ground truth of the i^{th} pixel, the brackets are the Iverson brackets such that $[P] = 1$ if P is true and $[P] = 0$ if P is false. These are the measures of the number of pixels that are labeled as part of the object and for which the ground truth has them marked as part of the object (TP), labeled as background and ground truth labels them as background (TN), labeled as object but ground truth labels them as background (FP), and labeled as background but ground truth marks them as part of the object (FN). The sum of these values is the total number of pixels in the image. Since CHM actually produces values that range from 0.0 to 1.0, the data is thresholded using an automatically calculated threshold from Ostu's method.

These four values compose the *confusion matrix* and can be used to define several other properties:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{TN + FP}$$

The accuracy, F-value, and G-mean metrics can be calculated using all these values:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Fvalue = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

$$Gmean = \sqrt{Recall \times Specificity}$$

All of these metrics range from 0.0 to 1.0 with 1.0 representing a perfect segmentation relative to the ground truth data. These metrics each tell different information. While accuracy is the simplest and most straight-forward, many organelles only represent a small number of the total pixels and therefore very high accuracy values are expected when most of the background is labeled correctly. However, this does not give a good indication of over-segmentation versus under-segmentation. The F-value provides a significantly more robust measure, and because of the underlying precision and recall values it uses it can give an indication of over- or under-segmentation. If the precision is high and recall is low, overall the image is over-segmented; while if the precision is low with high recall then under-segmentation is indicated. Small shifts in the boundaries of an object result in a balance of precision and recall. However, F-value is not symmetric to the positive and negative samples. The G-mean is used to address that issue and it is symmetric with respect to positive and negative classes.

These metrics work well for the results of CHM which outputs a binary classification. However, these do not work well with the results of GLIA since each region is given a label and it does not matter if the value of the label is different than the ground truth label value but only that the entire region is given a consistent label as in the ground truth. A pairwise pixel metric is defined to correct for this for each of the confusion matrix entries as defined by W.M. Rand (Rand, 1971):

$$TP_{pp} = \sum_{i} \sum_{j>i} \left[L_i = L_j \wedge G_i = G_j\right] \qquad FP_{pp} = \sum_{i} \sum_{j>i} \left[L_i = L_j \wedge G_i \neq G_j\right]$$

$$FN_{pp} = \sum_{i} \sum_{j>i} \left[L_i \neq L_j \wedge G_i = G_j\right] \qquad TN_{pp} = \sum_{i} \sum_{j>i} \left[L_i \neq L_j \wedge G_i \neq G_j\right]$$

This compares the labels of every pair of pixels to see if they match in the labeled image and the ground truth image. The TP and TN values count the number of accurately segmented pixel pairs. The FP counts the number of under-segmented pixel pairs and FN counts the number of over-segmented pixel pairs. The original Rand metric was calculated as the $Accuracy$ equation above but using the pairwise pixel values. The adapted Rand metric is $1.0 - Fvalue$ using the pairwise pixel values (Arganda-Carreras, Seung, Vishwanathan, & Berger, 2013). The G-mean value can also be used for the same reasons above but using the pairwise pixel values. For the project the G-mean value is used as it is symmetric with respect to positive and negative samples.

When comparing G-mean values, 1-tailed Student's T-test was used to determine significance. Depending on the variances of the values being compared, either a homoscedastic or heteroscedastic type is used.

5.6 Endoplasmic Reticulum Segmentation

ER are segmented using a basic threshold of the image, after removing segmented mitochondria and cell membranes. ER are dark in the EM images and the other structures that share the same gray levels are the cell membranes and mitochondria. Thus, dark pixels in the image after subtracting any pixel that is either a mitochondria or cell membrane are extremely likely to be ER. Thus, all gray values less than 65 (out of the range 0 to 255 for unsigned 8-bit images) are labeled as ER.

5.7 GeoEM

GLIA generates a cell label for each pixel in the image. These label images are converted to contours which circumscribe each unique label, an IMOD model, and then an IMOD polyhedral mesh as described in section 5.4. The volume of each segmented object is found using the command `imodinfo −c` and this information is used to select possible candidates for analysis with geoEM. The largest objects (but not the object representing the background) were extracted to new models with `imodextract`. The meshes from these models were then converted to the more portable format OBJ using the `imod2obj` program. Care must be taken as IMOD uses an unusual standard with the middle of the first voxel is located at $(0.5, 0.5, 0)$ instead of $(0.5, 0.5, 0.5)$. Additionally, this process has not yet accounted for the fact that the axial resolution is much larger than the lateral resolution. Fixing these problems can be done at once by transforming every vertex in the OBJ file by first translating it by $(0, 0, 0.5)$ and then scaling it by $(5.2, 5.2, 35)$ for the neuropil dataset. This was completed using a simple Python script.

At this point the mesh of an individual cell has been extracted into an OBJ file and properly translated and scaled. However, before it can be analyzed it must be repaired and refined. The refinement uses isotropic remeshing with a target edge length of 200 nm.

After the mesh is prepared then it is analyzed in 3D, along with the label images for the mitochondria or ER, using the geoEM program and the calculated values are saved along with a copy of the mesh that has each of the slices of the mesh colored according to its value. GeoEM uses 3 skeleton vertices per slice, giving a good balance between quality and consistency in the values.

6 Results

In the first few sections the results of PyCHM will be discussed. First the improvements to accuracy and the use of new features such as the Frangi filter and masking are examined. Then the improvements to the speed and memory usage of both training and testing of PyCHM are discussed. These sections are followed by the results of GLIA in automatic segmentation of neurons from dense neuropil and geoEM on manually and automatically segmented neurons with both mitochondria and ER organelle information.

6.1 CHM Accuracy

The data suggest that the new implementation of the CHM-LDNN algorithm in Python, PyCHM, increases the accuracy of the results over the previous MATLAB implementation. Even when using PyCHM with the MATLAB compatible set of options (i.e. no normalization and with the Gabor filter but without the Frangi filter), the G-mean values typically improved slightly (lysosomes without subsampling and mitochondria with or without subsampling significant at $p < 0.01$). This is can be attributed to the corrections in the various filters and the reduction of edge and seam effects. However, the new set of normalization and filter options can be used to improve the results even more for some datasets as shown in Table 6.1. When only looking at the MATLAB results, all datasets significantly benefitted from using subsampling (SS) as $p < 0.001$ except the nuclei dataset (average G-mean 0.9864 vs 0.9863 without and with subsampling respectively, $p = 0.46$). In fact, the other datasets averaged a 0.015 increase in average G-mean with subsampling. The atypical results for nuclei could perhaps be explained by the fact that nearly half of the pixels in the nuclei training dataset are labeled and thus close to balanced between labeled and unlabeled pixels.

PyCHM has several additional options to explore and the optimal set was found using the best average G-mean value after 3 runs then 7 additional runs were performed with the optimal set of options.

Table 6.1. CHM and PyCHM Accuracy on organelle datasets. Results are averaged from 10 training/testing runs. The optimal options were determined by using 3-10 runs with every combination of settings and selecting the set of options with the highest average G-mean value. The MATLAB version can optionally use subsampling (SS) but always uses no normalization and the Gabor filter. PyCHM has various normalization methods (none, min-max, mean-std, median-mad, or iqr), can optionally use subsampling (SS), and additional filters (Gabor and Frangi). Bold values indicate the larger G-mean value between MATLAB and Python versions, significant at $p < 0.005$.

Organelle	CHM Version	Optimal Options	Accuracy	F-value	G-mean
Lysosomes	MATLAB	none, SS, Gabor	0.996	0.554	0.948
	Python	min-max, Gabor, Frangi	0.958	0.162	**0.970**
Mitochondria	MATLAB	none, SS, Gabor	0.969	0.487	0.952
	Python	none, SS, Frangi	0.968	0.481	**0.955**
Nuclei	MATLAB	none, Gabor	0.984	0.975	0.986
	Python	none, SS	0.986	0.978	**0.988**
Nucleoli	MATLAB	none, SS, Gabor	0.992	0.868	0.968
	Python	none, SS, Frangi	0.989	0.835	**0.980**

All datasets except lysosomes performed best with no normalization, although these same datasets also gave the best results without the Gabor filter, which was the driving force behind adding normalization to the filters. The lysosome dataset instead did best with the min-max normalization along with using the Gabor filter. The same set of filters that did not use Gabor or normalization also preferred subsampling while lysosomes did not use subsampling. Like with the MATLAB version, use of subsampling with the nuclei dataset was not significantly significant (average G-mean 0.9879 vs 0.9880 without and with subsampling respectively, $p = 0.24$). The nuclei dataset was the only dataset that did not get the best results while using the Frangi filter, but the difference was not significant (average G-mean 0.9880 vs 0.9877 without and with the Frangi filter, $p = 0.089$).

6.1.1 Masking

The new masking feature of PyCHM was explored by using a mask to only train on the central 360×360 and 400×400 regions of the organelle datasets but still provide the entire training dataset so that the border pixels can be used to calculate the features for each pixel. The accuracy of these models was compared against cropped versions of the training data which prevents the use of this border region from being used for feature extraction which than uses a reflection of the image data along the borders.

The data in Table 6.2 indicates that masking has no effect in this situation for any of the datasets. Sometimes it produced worse results but always remained quite close to the cropping results. It is within the range of values seen by changing the size of the training dataset. The surprising result here is that with MATLAB the much smaller training sets can produce better results overall. For example, the average G-mean value continually improved for lysosomes and mitochondria as the dataset size was decreased. These datasets have the fewest labeled pixels in the training datasets, so the increase in labeling percentage provided by shrinking the dataset, bring the count of labelled and unlabeled pixels closer to balanced, is likely the reason they benefited from the smaller datasets. However, this effect does not occur with PyCHM in any of the explored cases.

Table 6.2. Masking usage on organelle datasets. The results of using the central 360×360 and 400×400 regions of the organelle datasets, either by cropping them before running training or using masking during training (which then includes the border data during feature extraction but not during the actual training). Values are averaged from 10 runs using the optimal settings per Table 6.1. Bold values indicate the larger G-mean value for an organelle and image size.

			$\mathbf{360 \times 360}$			$\mathbf{400 \times 400}$		
Organelle	**CHM**	**Central**	**Acc.**	**F-value**	**G-mean**	**Acc.**	**F-value**	**G-mean**
Lysosomes	MATLAB	Crop	0.995	0.481	0.962	0.996	0.530	0.953
	Python	Crop	0.981	0.208	**0.979**	0.972	0.165	**0.976**
		Mask	0.996	0.523	0.949	0.996	0.542	0.948
Mitochondria	MATLAB	Crop	0.969	0.488	**0.943**	0.971	0.499	0.940
	Python	Crop	0.969	0.488	0.942	0.971	0.502	0.943
		Mask	0.968	0.476	0.940	0.968	0.479	**0.944**
Nuclei	MATLAB	Crop	0.981	0.971	0.983	0.983	0.974	0.985
	Python	Crop	0.983	0.974	**0.985**	0.984	0.976	**0.986**
		Mask	0.980	0.970	0.983	0.981	0.971	0.984
Nucleoli	MATLAB	Crop	0.982	0.750	**0.982**	0.990	0.836	**0.972**
	Python	Crop	0.984	0.770	0.966	0.987	0.805	0.969
		Mask	0.982	0.741	0.966	0.984	0.773	0.969

Masking also greatly increases memory usage, thus creating a border using masking is not always optimal as it does not improve results and uses more memory. However, in other situations it may still be useful to mask out a border when the training set is not rectangular. This means that not every pixel within a sub-volume needs to be labelled, only the ones under the mask. This could be used to create a system where users only need to indicate small seed areas to get a training set started.

Masking was also attempted with the neuropil dataset by first segmenting the mitochondria and then masking them out before segmenting the cell membranes. During training masking was also used to ignore the mitochondria. When restricting the options to always use median-mad normalization, the best cell membrane detection was found using subsampling, the Frangi filter, and without the Gabor filter resulting in an average G-mean of 0.892 across 10 runs using the split training/testing neuropil dataset. When using masking, the best accuracy was also found when using subsampling, the Frangi filter, and without the Gabor filter. The mitochondria data used no subsampling or Gabor filter but did include the Frangi filter. The data strongly suggests that the masking process improved the G-mean ($p < 10^{-8}$); in fact the average G-mean was 0.916 across 10 runs – an increase of 0.026 over unmasked runs. This great improvement in accuracy comes at a cost however since two separate structures need to have training sets manually segmented and the training process needs to be run twice, once for each dataset. The testing phase is also twice as long since it will need to first be tested for one of the labels and followed by the second label.

The best mitochondria settings for masking actually segmented the mitochondria the worst according to the G-mean values. This may mean that it does best when the previous mask is a bit under-segmented but it also may be unique to this combination of labels. Until more evidence is found, if the first label is used for anything besides masking than additional training and testing may be required for the different purposes. Again, this increases the time and memory needed to perform the segmentation.

Due to the sheer number of settings possible, there are a number of combinations of settings which were not explored during this project.

6.2 CHM CPU and Memory Usage

The conversion of CHM from MATLAB to Python also brought significant changes in CPU and memory usage, typically showing major improvements for both. The training phase saw mild improvements (16% to 33% faster and up to 40% less memory usage, detailed in Table 6.3) but the testing phase saw major improvements ($10x$ to $52x$ faster and up to 46% less memory usage, detailed in Table 6.7). Most importantly, the testing improvements have brought the speed of CHM up to the speed of acquisition of images from SBFSEM when using common desktop hardware. The filters chosen during the training process effect both the training and testing speed and memory usage, with the Gabor filter having a very large negative impact on both.

6.2.1 Training

Even though the Python version of CHM does improve the speed of training and reduces the memory required for the training of a CHM model in most cases, it is still very expensive and requires specialized hardware such as having 90+ GiB of memory which is not common. Training of machine learning systems is notoriously expensive in terms of memory and time, so this is not an unexpected result.

Overall, PyCHM requires one fifth to one third less time than the original when only using one thread as shown in Table 6.3. The original MATLAB CHM benefitted more from additional threads, however even when it was given 4 threads it was still significantly slower than PyCHM with a single thread (10% to 22% faster). Additionally, both versions do not utilize symmetric multithreading well (e.g. Intel's Hyperthreading, shown when using 8 threads in Table 6.3).

Memory utilization results, also in Table 6.3, are a bit more mixed. When not utilizing subsampling, PyCHM needs only 60% of the memory required for the original CHM (although that is still 109 GiB). When using subsampling, the original CHM actually used less memory than PyCHM, by 8%. For both, the number of threads did not significantly influence the amount of memory used.

PyCHM also introduced several options including different filters and normalization of feature vectors whose efficiencies are compared in Table 6.4. The default normalization option, median-mad, is slightly more expensive in computation (1%) than the other methods. It is also 4% more memory intensive than no normalization (min-max normalization takes no significant extra memory or time over no normalization). Using iqr normalization requires nearly 10% more memory than when median-mad normalization is used. The new Frangi filter only adds a small amount of time and memory usage (3% and 2% respectively). On the other hand, removing the Gabor filter reduces the computational time by more than one third and the memory usage by one quarter. This means that unless the Gabor filter actually helps improve the accuracy of a particular model, which it rarely does in the tests performed, it should be removed as this reduces the time by hours and makes the training process usable on more machines.

Table 6.3. CHM training CPU and memory usage. Run with and without subsampling (SS). MATLAB CHM uses Gabor filter. PyCHM uses median-mad normalization and Gabor filter. Wall time is the real-world time that elapsed during the program run.

		Wall Time (hh::mm:ss)			**Max Memory Usage (GiB)**		
	Threads	**MATLAB**	**Python**	**Change**	**MATLAB**	**Python**	**Change**
w/o SS	1	20:16:37	16:06:00	-21%	179.55	108.47	-40%
	2	18:35:47	15:25:04	-17%	180.00	108.48	-40%
	4	17:46:17	14:54:43	-16%	180.02	108.52	-40%
	8	18:15:39	14:44:05	-19%	180.05	108.63	-40%
w/ SS	1	9:09:39	6:07:50	-33%	90.82	98.56	+9%
	2	8:09:16	5:56:40	-27%	91.32	98.57	+8%
	4	7:29:42	5:46:31	-23%	91.31	98.62	+8%
	8	7:23:00	5:41:50	-23%	91.25	98.72	+8%

In addition to dedicated training runs to measure the time and memory usage in a controlled environment, the timing of other training runs was performed but on shared and variable resources with training sizes that ranged from 368,640 pixels to over 20 GP with 1% to 46% of the pixels labeled. Nonetheless, the results are informative about the trends of the usage across varying training dataset sizes. Without subsampling, the memory usage is extremely linear with the number of pixels in the dataset and independent of the hardware being used as shown in Table 6.5. The Python version, PyCHM, is saving at least 39% in memory usage compared to the MATLAB version. It also shows that overall removal of the Gabor filter saves even more memory when not subsampling (about 30%) and that the Frangi filter only requires an additional 1% to 2% of memory.

The timing of the runs without subsampling is also quite linear (but with some small fluctuations depending on the percent of the pixels labeled) and shows that PyCHM is three to four times faster than the MATLAB version depending on the settings. Additionally, the trends confirm that the Gabor filter takes a large portion of the time (around 40%) and that the addition of the Frangi filter adds only a small amount of time (2% to 4%).

Table 6.4. PyCHM training filter and normalization CPU and memory usage. These use 4 threads, subsampling, and the default normalization (median-mad) and filters plus Gabor unless otherwise noted. Baseline is the default settings for comparison. Wall time is the real-world time that elapsed during the program run.

Parameters	Wall Time (hh::mm:ss)	Change	Memory (GiB)	Change
Baseline	5:46:31		98.62	
Filter: +frangi	5:56:35	+3%	100.25	+2%
Filter: -gabor	3:49:07	-34%	73.47	-25%
Normalization: none	5:41:39	-1%	94.34	-4%
Normalization: min-max	5:42:44	-1%	94.34	-4%
Normalization: mean-std	5:41:47	-1%	98.62	0%
Normalization: iqr	5:43:37	-1%	107.79	+9%

When subsampling is used the memory usage and times are not nearly as predictable and the percent of the pixels that are labeled has a much greater impact on the resource usage as shown in Table 6.6. The results indicate that when there are more pixels that are labeled in the dataset, more memory and time it required to train. Many of the observed trends of the non-subsampled training runs can also be seen here, albeit much less pronounced. One important observation is that even though PyCHM took more memory for the lysosome data studied in detail when subsampling, that is not always the case since the MATLAB CHM has a much steeper memory slope than Python. The approximate crossover point is around 8.3 MP, but this does not take into account percent of pixels labeled so individual datasets around there may be on either side.

Table 6.5. CHM training CPU and memory usage trends without subsampling. These use the default normalization (median-mad) for Python and the default filters plus the extra filters as noted.

		Memory Linear Regression			**Time Linear Regression**		
CHM Version	**Extra Filters**	**Slope (KiB/px)**	**Intercept (GiB)**	R^2	**Slope (ms/px)**	**Intercept (hours)**	R^2
MATLAB	Gabor	15.13	-506	0.99991	11.4	2.0	0.933
Python	Gabor, Frangi	9.23	-235	0.99999	4.1	13.4	0.989
	Gabor	9.12	-233	0.99999	4.0	14.1	0.990
	Frangi	6.41	-242	0.99998	2.9	9.0	0.986
		6.30	-244	0.99998	2.8	6.3	0.986

Table 6.6. CHM training CPU and memory usage trends with subsampling. These use the default normalization (median-mad) for Python and the default filters plus the extra filters as noted.

		Memory Linear Regression			**Time Linear Regression**		
CHM Version	**Extra Filters**	**Slope (KiB/px)**	**Intercept (GiB)**	R^2	**Slope (ms/px)**	**Intercept (hours)**	R^2
MATLAB	Gabor	11.12	-2.17	0.845	2.1	10.6	0.840
Python	Gabor, Frangi	6.26	36.42	0.858	1.5	3.1	0.797
	Gabor	6.15	35.98	0.857	1.4	3.1	0.743
	Frangi	5.10	22.94	0.918	1.1	1.9	0.801
		4.98	22.55	0.917	1.1	1.7	0.857

6.2.2 Testing

While training had modest improvements in both computational time and memory usage, testing is significantly more efficient using PyCHM as shown in Table 6.7. When using a single task, PyCHM can operate at 22 pixels per millisecond, which is 21x faster than the original CHM for 1 MP images and 15x faster for larger images. In these situations, PyCHM only requires 2.2 GiB and 3.3 GiB, which is 46% and 20% less than the MATLAB CHM. This is a massive reduction in resources required for testing.

The original CHM did not increase memory usage significantly when increasing the number of tasks due to the way the blocks were divided up and processed in parallel. However, the redesigned block processing in PyCHM requires significantly more memory as the number of tasks is increased, but this is required to eliminate edge effects along the seams between the blocks while maintaining high throughput. However, even for large images with large numbers of tasks, the amount of memory required is quite reasonable even for off-the-shelf computer hardware.

On the small 1 MP image, MATLAB barely improved with increased parallelism while PyCHM's performance scaled well with the increase in tasks. On the large 40 MP image, the MATLAB CHM did benefit significantly from the increased parallelism, however PyCHM had a corresponding increase in performance (until Hyperthreading was used at 8 tasks). With four tasks, PyCHM was able to process 74.2

Table 6.7. PyCHM testing CPU and memory usage. These use a model created with subsampling and the Gabor filter but all other settings as default. The 1 MP image is 1000x1000 pixels. The 40 MP image is 8000x5000 pixels.

		Speed (px/ms)			**Max Memory Usage (GiB)**		
	Tasks	**MATLAB**	**Python**	**Speedup**	**MATLAB**	**Python**	**Change**
1 MP	1	1.03	21.7	21x	4.11	2.21	-46%
	2	1.13	40.0	35x	4.51	4.15	-8%
	4	1.29	66.7	52x	4.49	7.93	77%
	8	1.28	66.7	52x	4.46	7.93	78%
40 MP	1	1.44	21.4	15x	4.20	3.34	-20%
	2	2.78	39.9	14x	4.53	5.48	21%
	4	5.19	74.2	14x	4.55	9.75	114%
	8	8.01	88.3	11x	4.54	18.26	302%

px/ms using less than 10 GiB (many common desktop computers have 16 GiB of RAM). SBFSEM machines have dwell times around 10-30 µs/px which is right around this speed of 13 µs/px. Thus, at this point it is possible to run PyCHM testing in real-time with the acquisition of images from a typical SBFSEM.

PyCHM supports multiple threads in addition to multiple tasks. When doubling the number of threads but keeping the number of tasks the same, the speed is increased by approximately 3% (ranging from 2.4% to 4.5%). This 3% increase in speed is accompanied by a 50-60% increase in CPU utilization. Thus, it is only slightly faster while utilizing significantly more CPU time. When utilizing simultaneous multithreading (e.g. hyperthreading), the extra threads decrease the performance. However, unlike with tasks, memory usage does not significantly change with the increase in the number of threads. So, if there are physical cores available, they should be used, even if the machine does not have extra memory.

The filters and normalization chosen during training effect the speed and memory usage for testing as well with results shown in Table 6.8. In general, the speed change is greater than with training while the memory change is smaller.

As was done with training, the testing runs were also performed over a wide range of dataset sizes, from 6144 pixels per image to 6.3 MP per image as shown in Figure 6.1. These results strongly suggest that PyCHM testing scales extremely well with the size of the data and that the number of tasks chosen is what primarily dictates the amount of memory required. When using 2 tasks, there is a linear

Table 6.8. PyCHM testing filter and normalization CPU and memory usage. These use models created using 4 threads, subsampling, and the default normalization (median-mad) and filters plus Gabor unless otherwise noted. Testing values averaged over 1, 2, and 4 tasks each with 1 thread.

Parameters	Wall Time Change	Memory Change
Filter: +frangi	+9%	+1%
Filter: -gabor	-40%	-18%
Normalization: none	-4%	0%
Normalization: min-max	-1%	0%
Normalization: mean-std	0%	0%
Normalization: iqr	-1%	0%

portion as the one and only task uses increasing resources until the block size is reached (512×512) at which point the growth slows down. However, since the second task is still working on increasingly more pixels as the image size increases there is still significant growth in GiB per px. Finally, once the second task is fully utilized as well, then the growth slows down considerably. This slow growth only adds 53 bytes per pixel which means that on an off-the-shelf machine with 16 GiB of RAM, a 225 MP image could easily be processed with 2 tasks.

Even though memory usage has a multi-linear relationship, timing is still a simple linear relationship when testing as shown in Table 6.9. These show that PyCHM is $12x$ to $24x$ faster than MATLAB's CHM even when using the more expensive median-mad normalization. Additionally, these trends show that PyCHM also have 1/10th the startup time then the MATLAB version has.

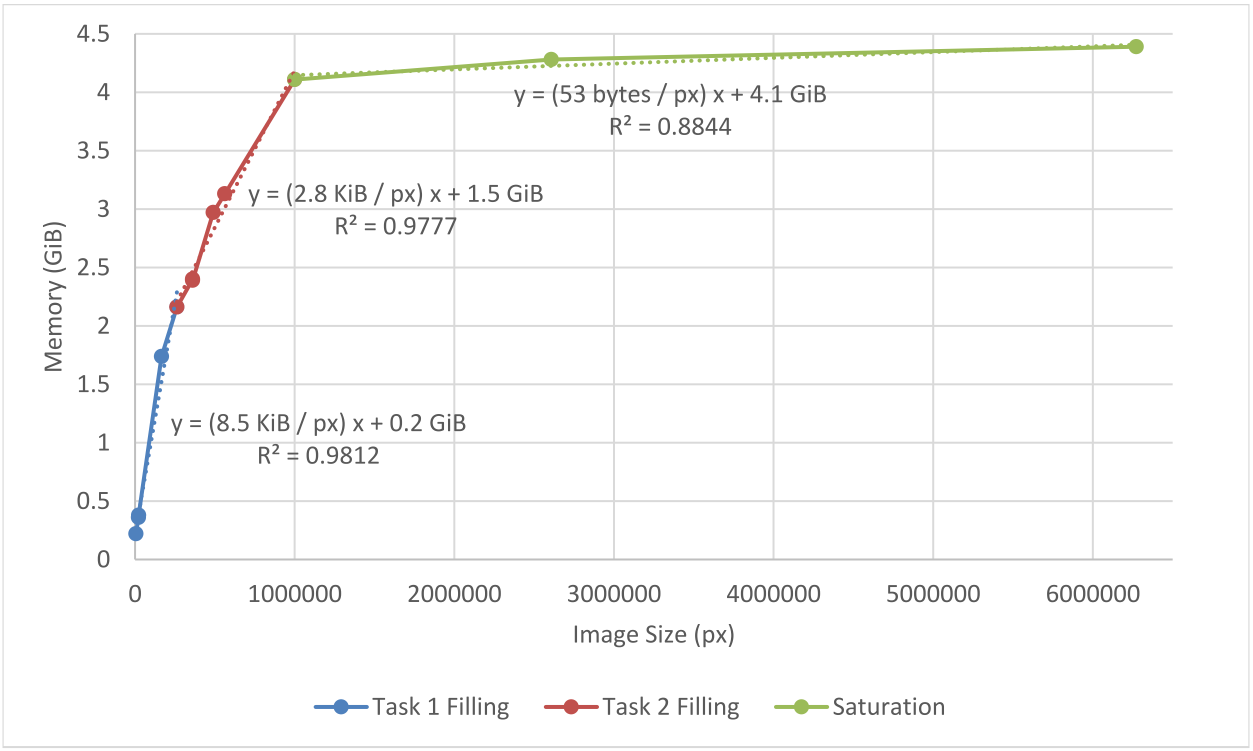

Figure 6.1. PyCHM testing filter and normalization CPU and memory usage. These use models created using 4 threads, subsampling, and the default normalization (median-mad) and filters unless otherwise noted. Testing values averaged over 1, 2, and 4 tasks each with 1 thread.

6.3 GLIA

Several different combinations of settings for GLIA and for the generation of cell membrane probability maps by CHM for use with GLIA were explored. The CHM trained models using cell membranes (without masking) that were created with median-mad normalization, subsampling, and the Frangi filter (but without the Gabor filter) were used. GLIA used the parameters $\sigma = 1.25$, $t_w = 0.05$, $t_{\alpha 1} = 50$, and $t_{\alpha 2} = 250$. These settings were chosen using finding settings that produced high G-mean values on the split neuropil training/testing dataset and through manual examination of the results when trained on the full combined neuropil dataset to confirm the presence of accurately segmented entire cells. When run on the split training/testing dataset, the G-mean of this combination of settings is 0.947 using the pairwise accuracy measurement. It is possible that changing the normalization setting, which was not experimented with for this part, could improve these results.

The trained CHM and GLIA models from the combined dataset were used to segment cells in other sub-regions of the neuropil dataset. These include a $750 \times 750 \times 110$ px region (61.9 MP, $3.9 \times 3.9 \times 3.85$ $\mu\mathrm{m}$, 58.6 $\mu\mathrm{m}^3$) and a $1615 \times 1615 \times 240$ px region (626 MP, $8.398 \times 8.398 \times 8.4$ $\mu\mathrm{m}$, 592 $\mu\mathrm{m}^3$). A 1.5 GP region was attempted as well but was unable to be completed with the resources available. The pre-merge step was unable to be completed within 48 hours on this dataset. The pre-merge step took 23.7 minutes on average to complete for the 626 MP dataset so it was surprising to find that it

Table 6.9. CHM testing time usage trends. Testing used models trained with the default normalization (median-mad) for Python and the default filters plus the extra filters as noted. The runs were performed with 2 tasks.

		Time Linear Regression		
CHM Version	**Extra Filters**	**Slope (μs/px)**	**Intercept (secs)**	R^2
MATLAB	Gabor	614	47.9	0.9996
Python	Gabor, Frangi	50	4.4	0.9978
	Gabor	47	5.3	0.9962
	Frangi	31	2.8	0.9983
		26	3.0	0.9972

took longer than 48 hours with a dataset only $2.4x$ as large. Upon further examination, the average time was misleading as the time ranged from 1.5 minutes to 2.8 hours. Other steps had much more consistent time usage, such as watershed requiring between 22.1 and 32.9 min with an average of 28.8 min for the 626 MP dataset. When performed on the 1.5 GP volume this step ended up taking about 1.25 hours which is about $2.4x$ the amount of time.

Additionally, the merge-order tree construction and boundary classifier feature extraction steps took a long and highly variable amount of time (both taking several hours) for the 626 MP dataset and likely would have followed the pre-merge pattern of dramatically increasing their time requirements when moving to the 1.5 GP volume which made that dataset completely out of reach with the available computational resources.

Due to GLIA learning the size of cells from the training set it is given, it is unable to segment cells outside of the trained range in any meaningful way. While the training set used was specifically designed to have both small and large cells in it so that GLIA would have a diverse set of inputs to learn from, the size of the training set itself limited how large and how many different examples could be present. Due to this, GLIA was unable to automatically segment any of the larger cells, including any of the major dendritic trunks within the dataset. It primarily segmented small processes through the datasets while classifying the large cells as part of the background. This problem is understandable because the original publication of the program only used datasets that had roughly equal sized training and testing sets were used.

The automatically segmented 626 MP volume in shown in Figure 6.2. The hole where a large dendrite goes through the volume is very obvious. However, the small dendrites are quite clear and well segmented with a few mistakes where one cell is split in two or two are combined into one.

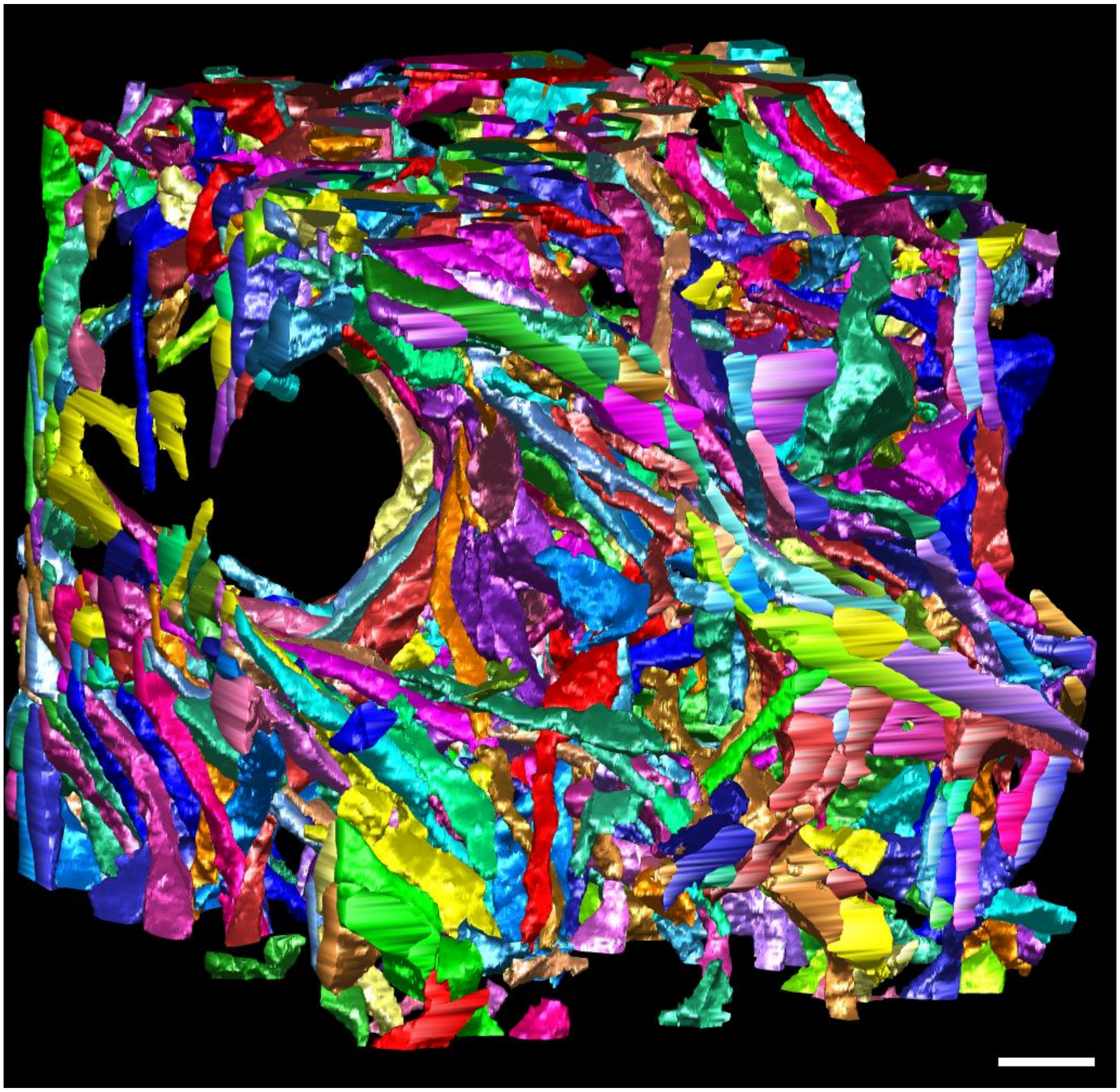

Figure 6.2. Automatically segmented neuropil volume. $592\ \mu\mathrm{m}^3$ (626 MP**)** volume of rat neuropil automatically segmented using PyCHM and GLIA using a manually segmented $19.1\ \mu\mathrm{m}^3$ (20.2 MP) training dataset. Scale bar is $1\ \mu\mathrm{m}$. The hole is because GLIA is unable to segment cells larger than any of the cells within the training set and it was marked as background. All objects smaller than $0.024\ \mu\mathrm{m}^3$ have been removed from the display.

6.4 GeoEM

Once a 3D neuron model is created with branches its geometrical properties can be analyzed with geoEM. Since no branching dendrites could be automatically segmented due to the limitations of GLIA, a major trunk of a dendrite and along with all its branches within the entire dataset were manually segmented for use with geoEM. This neuron model, including the partial soma, has a volume of $3537\ \mu\mathrm{m}^3$ and a surface area of $2241\ \mu\mathrm{m}^2$ after repairing and refining. The surface-area-to-volume ratio (SVR) is $0.63\ \mu\mathrm{m}^{-1}$ which is less than 1 and being dominated by the large volume and low surface area of the soma. Since the soma is not needed for the analysis of the dendritic branching and can cause issues with the analysis, most of the soma was removed from the model. With most of the soma removed, the model has a volume of $1024\ \mu\mathrm{m}^3$ and a surface area of $1299\ \mu\mathrm{m}^2$ which gives an SVR of $1.27\ \mu\mathrm{m}^{-1}$. Overall, the isotropic remeshing with a target edge length of $0.2\ \mu\mathrm{m}$ resulted in an average length of $0.18\ \mu\mathrm{m}$ and the final model had $94{,}262$ faces, $282{,}786$ edges, and $47{,}133$ vertices. The neuron has 3 branches coming off of the main trunk and two of those branches form additional branches. Sometimes geoEM mistakes the more pronounced dendritic spines as branches as well.

The mitochondria were automatically segmented with PyCHM using median-mad normalization, subsampling, and both the Gabor and Frangi filters. When run on the split training/testing dataset these settings achieved an average G-mean of 0.962. The ER were segmented by masking out the mitochondria and selecting everything with a gray value less than 65 out of 255. Since there was no cell membrane segmentation of this data, some of those may be misclassified as ER. However, they should all be outside of the model given and thus should have little influence on the results.

The density of both mitochondria and ER is relatively high near the soma and tapers off slightly towards the distal tips as shown in Figure 6.3 and Figure 6.4. There are local regions that have increased or decreased densities but no obvious correlation with distance as shown in Figure 6.5 and Figure 6.6.

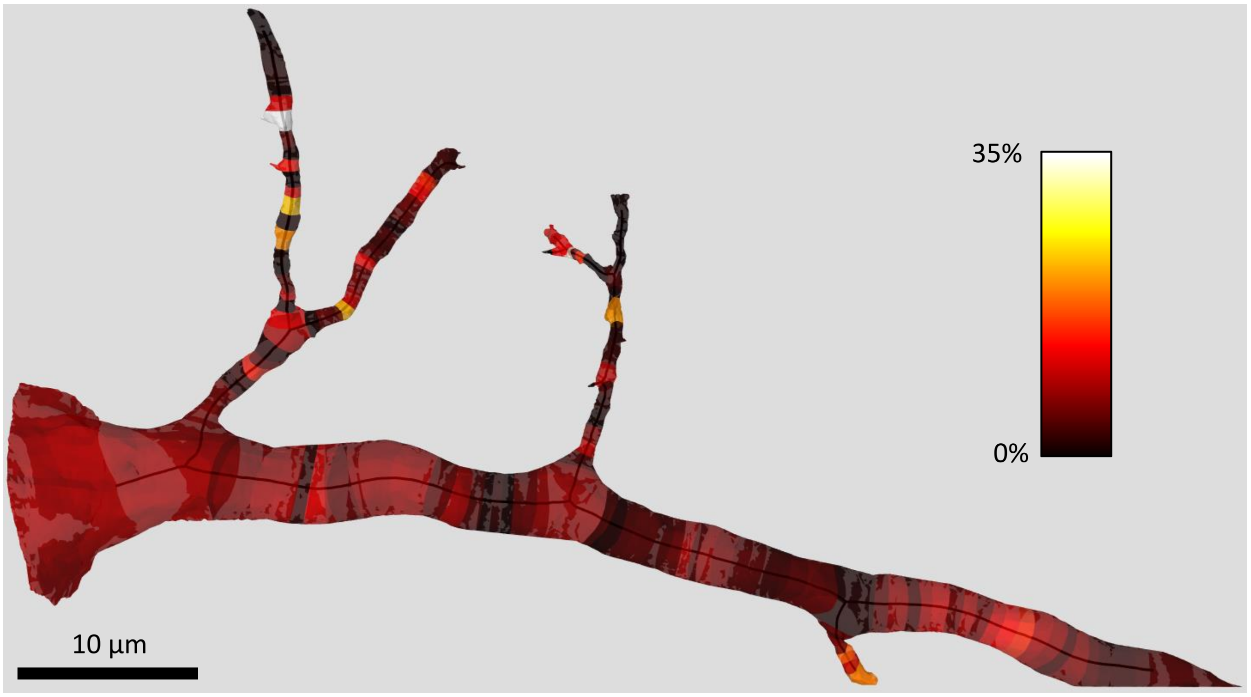

Figure 6.3. Density of mitochondria in a neuron. Shown as a percent of the local volume. The soma of the neuron is to the left in the image. The main trunk continues out of the dataset to the right and below. The skeleton is shown inside the mesh as a black line.

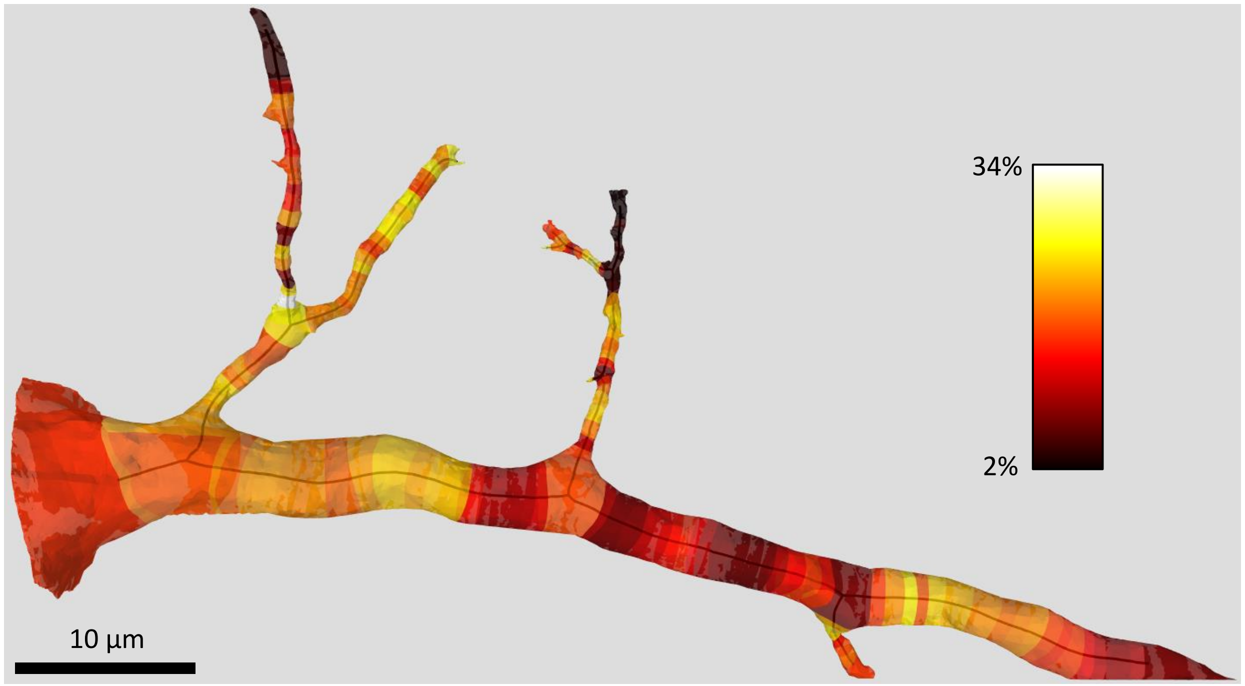

Figure 6.4. Density of ER across a neuron. Shown as a percent of the local volume. The soma of the neuron is to the left in the image. The main trunk continues out of the dataset to the right and below. The skeleton is shown inside the mesh as a black line.

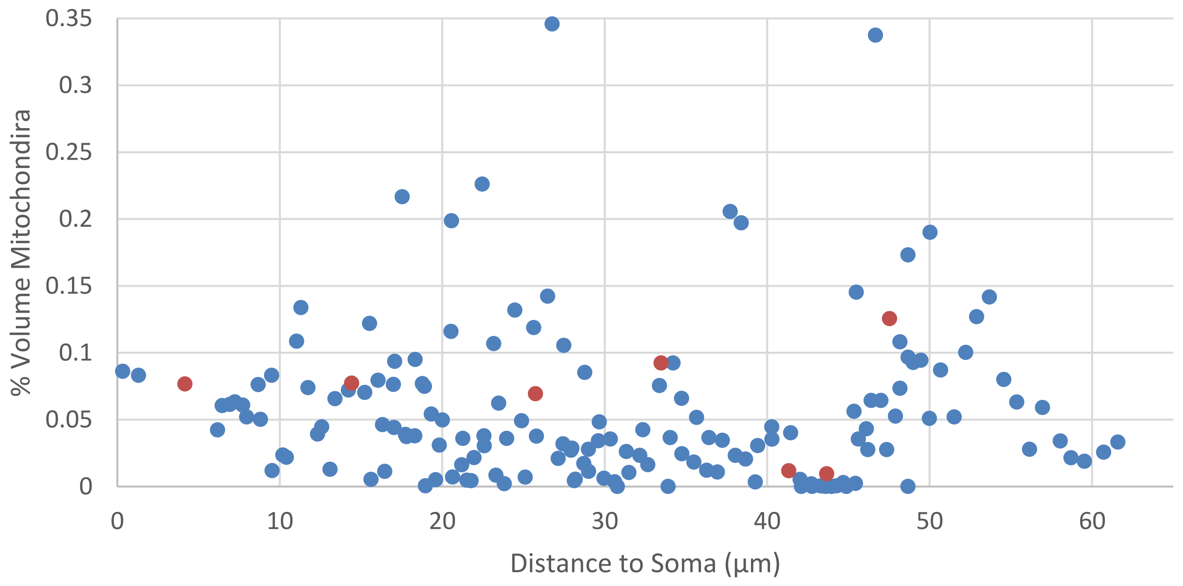

Figure 6.5. Density of mitochondria relative to branch point locations. The local volume of mitochondria changes with the distance to the soma. Data from 156 slices in one partially segmented dendritic tree. Red circles indicate branch points.

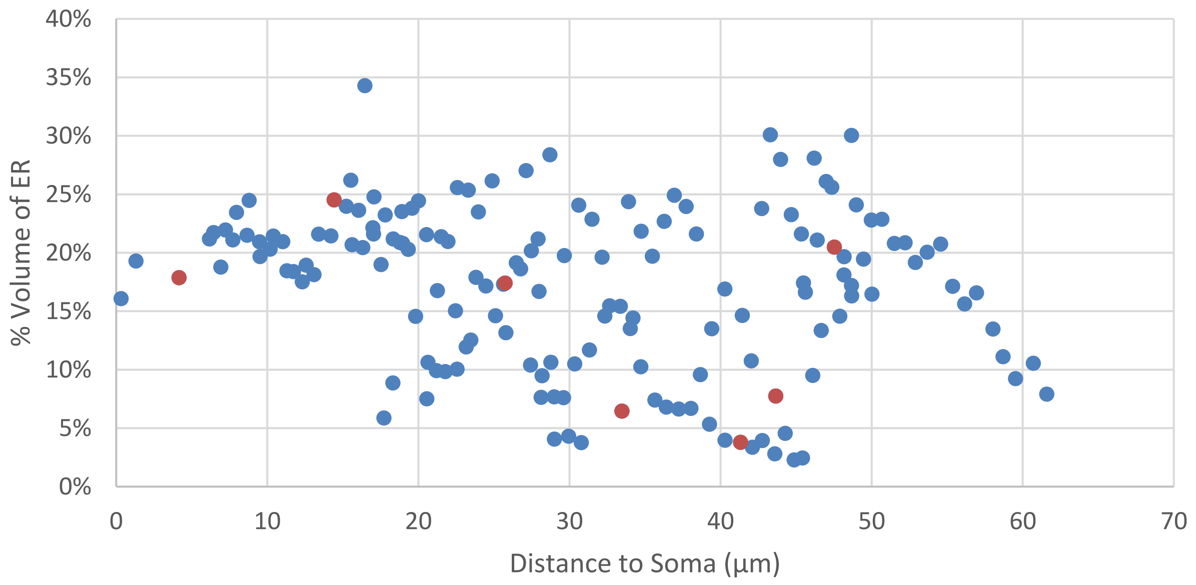

Figure 6.6. Density of ER relative to branch point locations. The local volume of mitochondria changes with the distance to the soma. Data from 156 slices in one partially segmented dendritic tree. Red circles indicate branch points.

7 Future Directions

Based on the outcome of this project, several suggestions for future research, both in technique and application, can be made. This chapter begins with how the tools developed here can be used for several different neuroscience applications. This is followed by a look at the many possible avenues for improvement in the automatic segmentation pipeline which would likely improve the efficiency, accuracy, and usability for the average neuroscientist.

7.1 Neuroscience

The primary goal of this project was to assess and develop viable ways to automatically segment and quantitatively analyze high-resolution EM images of neuropil. Towards that goal, a few basic metrics of neuron geometry and organelle localization were assessed in a $1024\ \mu\text{m}^3$ neuronal dendrite volume from the hippocampus CA1 region of a rat brain. Now that the tools exist they need to be more rigorously verified and eventually they can be used to gather more detailed information about many other neurological systems.

One immediate application is to use these tools to study the morphology around branch points within the hippocampus and the distribution of mitochondria and ER. A larger sample size than the one single neuron used in this project needs to be investigated to determine the trends. Additionally, animals that developed in different environments, such as enriched and impoverished environment, can be studied to look for differences in the morphology and organelle distribution. It is already known that there are electrochemical differences within the neurons in the animals from different developmental backgrounds (Losonczy, Makara, & Magee, 2008), the next step is to discover what is causing the differences.

More detailed examination of the effect of taper on the propagation of signals through dendritic trees needs to be done. This has been accomplished at the light microscopy level, but with using non-automated techniques dependent on human interpretation of the data (Ferrante, Migliore, & Ascoli, 2013). However, the methods developed here could be used to make a more robust analysis of the taper property of the branches.

Both of the studies described above were originally done with light microscopy where the samples remained alive and exhibited behaviors. That is not possible with samples prepared for EM. However, a correlated light and electron microscopy (CLEM) study could be performed that identifies branch points with unique electrophysiological property using light microscopy and then utilizes EM to study the underlying structure of these branch points.

While the metrics that geoEM currently provides are useful, several additional metrics would be required to study the above. The two metrics required for the above studies would be: 1) calculating the taper of each slice (Ferrante, Migliore, & Ascoli, 2013) and 2) the location of the organelles relative to the cell membrane which would indicate subsurface cisternae and other tethered structures participating in signal propagation along the cell membrane or organelles near the medial axis of the dendrites which are more likely for transport of enzymes throughout the dendrites.

7.2 Dendritic Tree Simulations

One longer term goal is to modify geoEM to for the creation of simulation models that can be high-resolution and high-accuracy in regions of interest around the branch point while also providing extremely efficient calculations for long-range single down branches. This could be done utilizing a combination of the simulation environments MCell and NEURON. MCell is a Monte Carlo simulation environment for cellular environments, simulating individual ion channels, buffer proteins, and even the ions themselves (Kerr, et al., 2008). While MCell has an impressive ability to create highly realistic simulations, this comes at the cost of long computational times and entire dendritic trees are too large for simulation currently. On the other hand, NEURON simulates at the level of compartmental Hodgkin-Huxley-style neurons and can efficiently simulate the electrophysiology of entire complex dendritic branching structures (Carnevale & Hines, 2006). However, it lacks the ability to efficiently or accurately simulate detailed interactions of buffers, kinases, and ion channels. Since we are interested in the branch points, the MCell environment could be used to simulate the dynamics occurring within the immediate vicinity of a branch point while the NEURON environment could be used to simulate the electrical properties along the branches, connecting the branch points together.

One challenge in bridging these two environments is how they treat the space that they are simulating. MCell, being high detailed, uses arbitrary 3D polyhedral meshes like those generated by segmenting EM data. NEURON operates at a lower level of detail and thus cannot handle meshes but instead works with a connected series of truncated cones (which are just cylinders but can have a taper with them, i.e. different size ends). This is where the geoEM tool comes into play since it converts a 3D mesh into a series of slices which are close approximations of truncated cones. The same 3D mesh could be divided up with the branch point slices being simulated with MCell and the other slices being approximated with truncated cones and simulated using NEURON.

7.3 Segmentation Improvements

Each step in the process of converting the raw data from the microscope to getting quantitative data about the geometry of dendrites and location of organelles has room for further exploration. Here we will go through each of those steps describing future directions and improvements for each of them.

7.3.1 Pre-Processing

All of the microscopy data is pre-processed by using exact histogram equalization to increase the contrast in the images. However, exact histogram equalization methods suffer from the inability to use the same transform across multiple images. Thus separate slices from a single image volume will undergo slightly different transformations meaning that that the automatic segmentation algorithms may see them differently. It may be possible to lift this limitation of exact histogram equalization with an offline algorithm to establish the strict ordering of all pixels across large amounts of data and separate images.

Although it would be preferred to have all images in a 3D image stack undergo the same exact transformation this is not optimal because of technical issues with acquiring SBFSEM. In SEM the sample builds up a negative charge over time due to the continual bombardment by electrons. This charge causes future electrons to behave not as consistently, called electron beam drift. This effect is most pronounced in SBFSEM since the sample undergoes much more prolonged exposure to the electron beam resulting in noticeable differences in contrast between the first and last slices of an image stack. This charge build-up does affect individual SEM acquisitions as well as individual slices from a 3D stack of images acquired in SBFSEM, just to a much lesser extent resulting in a slightly different contrast level in each corner of the image.

There are several physical methods being actively used and some being developed to mitigate electron beam drift so that that the raw images themselves do not have as pronounced of a contrast difference, however they are not perfect (Bouwer, et al., 2016). The problem could be partially overcome computationally by using a technique called adaptive histogram equalization (AHE) (Hummel, 1977) or the extension of it: contrast limited adaptive histogram equalization (CLAHE) (Zuiderveld, 1994). In AHE the source histogram is calculated per-pixel based on its neighborhood. The size of the neighborhood determines at what scale contrast is enhanced and at what scales contrast is reduced. At scales smaller than the neighborhood size contrast is enhanced while at larger scales it is reduced. If too small of a scale is chosen AHE has the tendency to greatly amplify noise, the ultimate in increasing contrast at a small scale. At large enough scales it becomes nearly equivalent to traditional histogram equalization. The CLAHE extension also reduces the amplification of noise by limiting the amount of contrast enhancement that is possible in any neighborhood.

To make AHE more computationally feasible for large images the image is separated into tiles the size of the neighborhoods and the histogram for each tile is calculated. Then a linear interpolation of neighboring histograms is performed so that a source histogram can be approximated for every pixel in the image. This source histogram is then used to calculate the linear transform for each pixel.

For a SBFSEM image data stack where we are concerned about the overall drift of the contrast and not correcting for local changes in the contrast the neighborhoods, a natural choice would be each 1/8 of the overall image cube, each neighborhood being one of the corners of the dataset. The histogram for each would be calculated and then interpolated for interior points in the cube. This would then be accounting for a bilinear drift in the plane of each of the image slices and bilinear drift across slices. With neighborhoods this large there is essentially no risk of amplifying noise in the image or being biased significantly by local features. Additionally, the memory requirements for storing the complete transformation would only be 8 times that of traditional histogram equalization.

One drawback of AHE and CLAHE is that it can only be used with traditional histogram equalization at the moment. However, adapting exact histogram equalization to support offline calculation of the transformation as discussed previously could possibly be used with exact histogram equalization as well. The linear interpolations would be of the cutoffs used between histogram bins instead of the coefficients of the linear transformation.

All-in-all the ability to use a single exact histogram equalization transformation across many slices in a stack of images would increase the reliability of the data by homogenizing local regions. Additionally, being able to use the AHE or CLAHE technique in combination with exact histogram equalization will allow increasing the contrast in the image without reducing the amount of information.

7.3.2 CHM-LDNN

The core of this entire automatic segmentation is the CHM-LDNN algorithm which performs the classification of pixels based on their features. One major improvement to increase the speed of CHM would be to make several of the filters and the final evaluation of the model utilize GPU processing through CUDA or OpenCL. Many of the filters have had their memory requirements reduced which makes them more suitable for calculation on GPUs which frequently have less direct access to memory than the CPU. Additionally, since PyCHM has no tiling effects when breaking a larger image into smaller pieces the tiles could be automatically sized to work with the memory available on the GPU.

The filters that currently take the longest to run are Gabor, SIFT Flow, HOG, and Frangi so they would be the first targets for porting to the GPU. Gabor is primarily based on Fourier transformations which have dedicated libraries for both CUDA (cuFFT) and OpenCL (clFFT) so conversion to GPU would be a relatively straight forward. SIFT Flow, HOG, and Frangi filters operate on the image in small local neighborhoods and thus could be made to work efficiently on a GPU as well. The other filters could be kept on the CPU and run in parallel with the computations on the GPU.

The final evaluation of the model primarily involves a matrix multiplication, element-wise operations, and sum-reduction which are all very well suited to being calculated on a GPU and are well-studied. The problem with this however is that CHM-testing requires the most memory during the model evaluation and thus may not be suitable for running on the GPU due to memory bandwidth speed and limited available GPU memory. The training process is very memory intensive and is not easily parallelized and thus would not be a good target for porting to the GPU at the moment.

Additional improvements would be in adding additional filters to produce features that give the learning process more information about different types of textures and objects thus improving the results for a wide variety of structures to be segmented. For example, Radon-like features have been shown to be able to highlight membranes in EM images (Ritwik, Vázquez-Reina, & Pfister, 2010; Seyedhosseini, et al., 2011). The Gabor filter could be replaced by the log-Gabor filter which has been shown to perform better in several situations and does not suffer from the same DC problem that the standard Gabor filter does (Field, 1987). If it is not replaced then the Gabor filter can be likely improved by saving additional information about the complex-space function and removal of the larger kernels as those will be automatically captured in higher levels of CHM. Likewise, the Frangi filter may also benefit from limiting the number of scales it works at during each level. The SIFT Flow filter was never optimized or updated in PyCHM and could use an overhaul. Finally, adding features that used color information, or any multi-channel data such as multicolor EM (Adams, et al., 2016), if available, instead of single-channel grayscale images would improve the ability to work on color images and having filters that used 3D information would allow for improvements in 3D volume datasets.

The filter normalization process could benefit from additional improvements as well. Several more advanced normalization methods may be used that could improve the data being given to LDNN, such as using Hampel tanh-estimators (Hampel, 1986) or double sigmoid functions (Cappelli, Maio, & Maltoni, 2000) which have been shown to improve results in machine learning systems. Another change might be to increase the amount of information about the expected distribution of each feature so that the normalization procedure can use this to better perform the normalizations. For the moment, the only information given to LDNN about the expected distributions of an individual filter is whether it follows a roughly normal distribution, but additional information would make it possible to better adjust it better to give LDNN the best opportunity to separate positive from negative samples.

In the original CHM paper, the authors showed a proof-of-concept of a multi-labeling system that essentially ran CHM in parallel for each label but shared information between the separate labels between each level as extra contextual features when working with the Stanford Background dataset (Seyedhosseini & Tasdizen, 2015). While this only provided a small increase in quality (less than one percentage point increase of G-mean) for that model, other datasets may benefit from it even more (such as the Corel subset and Sowerby subset datasets (He, Zemel, & Carreira-Perpinan, 2004)). It could be implemented for training by significantly increasing memory usage (for n labels using n-times as much memory) but run in parallel (which would be great since the main part of training only runs single-threaded) or to use just a small amount of additional memory but take significantly more time (for n labels using n-times as much time). Additionally, it could be done in such a way to balance the two (memory and time) for the current system. Testing time and memory would likely only slightly increase regardless of the number of labels.

The current target values of the training process of 0.1 and 0.9 were arbitrarily chosen in the original design of CHM. Preliminary results show that the target values should be pushed closer to 0 and 1 respectively, possibly even to 0.01 and 0.99 since after testing there are peaks at the target values with a steep drop-off towards 0 and 1 with only 0.0005% of values between 0 and 0.01 and 0.99 and 1. This means that the possible range of outputs is not being saturated and never reached and thus the targets could be pushed further towards the extremes (LeCun, Bottou, Orr, & Müller, 1998). This may improve the results as more of the output range would be utilized and, since the k-means clustering is unaware of these target values, the initial weights calculated will be closer to the actual final weights resulting in starting closer to the minimum.

Another area for optimization is in the gradient descent procedure of the LDNN training process. The current parameters were chosen since they work but are not necessarily optimal. Further testing should be able to determine if the learning rate can be increased (which would reduce the error of the model in fewer iterations). One possibility is an annealed learning rate which is when the rate is initially very high and after each iteration the learning rate is decreased. This allows faster convergence combined with more refined iterations at the end to find a local optimum (Bach & Moulines, 2011). Likewise tweaking of the momentum may prove to be beneficial, such as using the current momentum of 0.5 at the start and increasing it to above 0.9 after the initial learning has stabilized (Sutskever, Martens, Dahl, & Hinton, 2013).

Both of those will likely allow faster convergence to a local optimum but may increase the susceptibility to overfitting. However, these could be combined with the dropout method to decrease overfitting issues, although using dropout for the entire training of a single phase results in a very low reduction in error regardless of the number of iterations performed (Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 2012). An optimal situation may be to start out using the batched stochastic gradient for several iterations with a decreasing learning rate and increasing momentum followed by a non-batched (or very small batches such as 10-25 samples per patch) iterations using dropout while still modifying the learning rate and momentum.

Other gradient descent methods that use more advanced learning rate and momentum schedules can be tried as well, such as AdaGrad (Duchi, Hazan, & Singer, 2011), ADADELTA (Zeiler, 2012), or RMSProp (Tieleman & Hinton, 2012).

Another aspect that could be further explored is the use of subsampling. For some organelles, it was observed that subsampling greatly improves results and in other cases greatly hampers the results. In some cases subsampling increases the variability of the results and sometimes it reduces the variability of the results. It is possible that these changes are dependent on the ratio of positive samples to negative samples, or relative rarity of one of the sets of samples to the other leading to algorithmic bias for one of the sets over the other. The subsampling is correcting this in some cases by bringing the number of samples of each closer together or even equal at the expense of losing information. Instead of subsampling another approach might improve results even more, such as balancing by over-sampling the minority set or under-sampling the majority set (Kubat & Matwin, 1997; Japkowicz, 2000; Lewis & Catlett, 1994; Ling & Li, 1998) or blending the two as is done with SMOTE (Synthetic Minority Oversampling Technique) by Chawla *et al* (2002).

7.3.3 Post-Processing

Currently the post-processing of data from CHM is either a binary thresholding using Otsu's method to automatically determine the threshold or a ternary thresholding using a 3-level Otsu's method followed by hysteresis thresholding to get the binary classification. If the histogram of the data is bimodal then binary thresholding should be used and if the data has more than 2 modes then the ternary plus hysteresis thresholding should be used. At this point the user responsible for the data must look at it themselves and make this determination. A major improvement would be to develop a tool to perform the determination automatically. This could possibly be performed using a variation of Otsu's method as well to find out if a 2-level or 3-level division fits better with the data.

The post-processing performed after CHM to prepare the data for use by GLIA by stretching it so that the peaks were not near 0.1 and 0.9 but instead closer to 0.0 and 1.0. Alternative approaches to the manipulation could be explored, such as reflecting values outside the range. However, the real goal would be to have CHM produce data that was already scaled appropriately as the data from 0.0 to 0.1 and from 0.9 to 1.0 is ambiguous.

7.3.4 GLIA

Since GLIA currently uses LDNN as the machine-learning algorithm, thus suggested directions above are all relevant here. Some changes already explored in LDNN for this project such as feature normalization, could be explored for GLIA, while conversely some improvements to GLIA (balancing of samples) could be implemented directly in LDNN.

While this project explored some of the parameters of GLIA, there are significantly more parameters that can be adjusted with possibly large changes in the results. Like CHM, additional region filters could be added to help the algorithm better identify when to merge regions. One important piece of information that is generated by geoEM which could help GLIA form continuous dendritic trees is to use the surface area to volume ratio over slices to decide if something is actually neuron-like.

Finally, with improvements in efficiency implemented elsewhere in this project, the highest remaining barrier to segmenting large datasets (besides initial human segmentation for training) is GLIA's inability to efficiently segment large 3D volumes. In particular, the pre-merging and boundary classification feature extraction steps do not scale well with the size of the volume. When run on a volume that is $2417 \times 2595 \times 240$ (1.5 GP) the pre-merge step alone took longer than 48 hours when run on the XSEDE Comet cluster which is the time limit for processes on that cluster.

7.3.5 GeoEM

GeoEM needs several more refinements so that it can be used by the general scientist. Foremost among these are the need to be more robust in the face of difficult-to-use meshes. As discussed earlier in section 7.1, additional metrics can be added to the program. The program can be adapted to generate models that bridge the MCell and NEURON simulation environments as discussed in section 7.2.

7.3.6 Image Processing Pipeline

The current set of tools available with pysegtools are only accessible via Python scripts or the `imstack` command line program. CHM, GLIA, and geoEM are also only accessible via the command line.

Even though all of these programs have been simplified and made more usable, the general researcher will not want to go into the command line to use these tools. Giving these tools a graphical user interface or a website portal where jobs can be submitted would be a major benefit to the general scientific community, making these tools available to all researchers.

Additionally, any reduction in the amount of training data that needs to be manually segmented by an export required would greatly expedite the process. The ultimate goal of the automatic segmentation would be to have all data coming from a microscope automatically segmented with minimal user intervention, something as simple as indicating a single instance of an organelle on the first slice of data. This would require being able to transfer knowledge from previously trained models to a new dataset that may have slightly different image properties.

Appendix A: Division of Tiles into Groups by PyCHM

This appendix describes how tiles are divided into groups by PyCHM when requested for parallel execution so that each group performs roughly equal amounts of work.

The algorithm for dividing up the groups always makes each group rectangular as any other shape increases the number of neighboring blocks needed for a particular arrangement. The groups are arranged in rows with each group in the same row having the same height. The extra blocks that are required for most rows are the blocks above and below the row (so $2W$ where W is the image width in blocks), the regions in between groups within a single row but not at the start or end of the row (so $2(g_i - 1)h_i$ where g_i is the number of groups in the ith row and h_i is the height of the ith row), and the kitty-corner corners for all corners not touching the edge of the image (so $4(g_i - 1)$). The first and last row are only different in that they have do not have a top or bottom set of blocks so their extra work is less by $W + 2(g_i - 1)$ each. The overall extra computational work for an arrangement with R rows is:

$$C_{Extra} = \sum_{i=1}^{R} \big(2W + 2(g_i - 1)h_i + 4(g_i - 1)\big) - (2W + g_1 + g_R - 4)$$
$$= 2\left((W - 2)(R - 1) - H + 2G - g_1 - g_R + \sum_{i=1}^{R} g_i h_i\right)$$

Besides minimizing the amount of extra work being performed all groups should be performing roughly the same amount of work so that the distribution is meaningful. The amount of work any individual group has to do is its area plus any extra work it must perform. The amount of computational work that the group in the ith row and jth with width w_{ij} column requires is:

$$C_{ij} = h_i w_{ij} + h_i([j \neq 1] + [j \neq g_i]) + w_{ij}([i \neq 1] + [i \neq R])$$
$$+ ([i \neq 1] + [i \neq R])([j \neq 1] + [j \neq g_i])$$
$$= (h_i + [i \neq 1] + [i \neq R])\big(w_{ij} + [j \neq 1] + [j \neq g_i]\big)$$

where $[S]$ is the Iverson bracket evaluating to 1 if S is true and 0 otherwise. A single row then requires the following amount of computational work:

$$C_i = \sum_{j=1}^{g_i} C_{ij} = (h_i + [i \neq 1] + [i \neq R])(W + 2(g_i - 1))$$

This can then be used to calculate the computational work needed for the entire arrangement:

$$C_{Total} = \sum_{i=1}^{R} C_i = HW + 2\left((W-2)(R-1) - H + 2G - g_1 - g_R + \sum_{i=1}^{R} g_i h_i\right) = HW + C_{Extra}$$

and thus the average computational work for any group is $C_{Avg} = {C_{Total}}/{G}$. We want to minimize both the total amount of extra computational work along with the sum of the absolute differences of each group's computational work from the average to encourage an equal workload. We can minimize the sum of these two factors and get the desired results. Overall, we want to minimize:

$$M = C_{Extra} + \sum_{i=1}^{R}\sum_{j=1}^{g_i}\left|C_{ij} - C_{Avg}\right|$$

When setting up the groups in a row we make it so all groups in the same row have approximately the same width since this leads to approximately the same amount of work for each group in that row. Some groups may be an extra block wide in cases where the image width in blocks is not evenly divisible by the number of groups in the row. It is optimal to place wider groups at the ends if possible as those groups already have lower workloads due to being on the edges of the image. We can combine this information with the extra of blocks needed at boundaries between groups in the horizontal direction within a row to get the total extra blocks in horizontal direction that must be computed for any group:

$$\begin{aligned} n_{ij} = {} & [j \neq 1] + [j \neq g_i] + [j = 1][W \bmod g_i \geq 1] + [j \neq 1][j = g_i][W \bmod g_i \geq 2] \\ & + [j \neq 1][j \neq g_i][W \bmod g_i \geq j + 1] \end{aligned}$$

This allows us to redefine C_{ij} as follows:

$$C_{ij} = (h_i + [i \neq 1] + [i \neq R])\left(\left\lfloor\frac{W}{g_i}\right\rfloor + n_{ij}\right)$$

We can then use this to remove the summation over j from the formula for M:

$$M_i = \sum_{j=1}^{g_i}\left|C_{ij} - C_{Avg}\right| = \sum_{j=1}^{g_i}\left|(h_i + [i \neq 1] + [i \neq R])\left(\left\lfloor\frac{W}{g_i}\right\rfloor + n_{ij}\right) - C_{Avg}\right|$$

The special case when $g_i = 1$ should be separated out as it simplifies the math significantly, resulting in:

$$M_i = \begin{cases} |e_i| & \text{if } g_i = 1 \\ \max(2 - r_i, 0)\left|e_i + \widehat{h_i}\right| + (g_i - |r_i - 2|)\left|e_i + 2\widehat{h_i}\right| + \max(r_i - 2,0)\left|e_i + 3\widehat{h_i}\right| & otherwise \end{cases}$$

where $\widehat{h_i} = h_i + [i \neq 1] + [i \neq R]$, $e_i = \widehat{h_i}\left\lfloor\frac{W}{g_i}\right\rfloor - C_{Avg}$, and $r_i = W \bmod g_i$. The overall metric we are now wanting to minimize is:

$$M = C_{Extra} + \sum_{i=1}^{R} M_i$$

This equation is not too intensive, with the calculation for C_{Extra} being conserved between each of the M_i calculations (as the major component of the C_{Avg} calculation), many of the calculations for M_i being reusable for a single i, and does not have any nested summations.

Now we must generate arrangements to evaluate the metric and minimize it. The metric requires the set of g_i values and the set of h_i values. The algorithm used here is a simple brute-force approach which generates each possible arrangement to get the g_i values but then calculates the h_i values so they are as close to optimal as possible for that set of g_i values.

The g_i values could be generated using the stars-and-bars theorem (Feller, 1950) but this would result in $O(2^G)$ different arrangements. Many of those arrangements are duplicates (e.g. $\{1,3\}$ and $\{3,1\}$ are equivalent). Restricting arrangements to only those for which the first row has at most the same groups as the last row and that all middle rows are weakly increasing in their number of groups reduces this to being a polynomial-order number of arrangements. This is described by the following rules:

$$g_i \leq g_{i+1} \forall i \in [2, R-2]$$

$$g_1 \leq g_R$$

The reason the first and last row must be handled separately from the middle rows is that those groups have a different number of neighboring blocks that need to be dealt with. The generation of the groups from these rules can be done with a simple recursive algorithm.

Once the groups are obtained the heights are needed. This can be done in a similar manner to the widths of the groups by first determining a height for each row that is possibly one less than it should be and then adding on one to a subset of the rows until the sum of the heights equals the actual height. The approximate height of each row is as:

$$\widetilde{h_i} = \left\lfloor \frac{Hg_i}{G} \right\rfloor$$

Then $H - \sum \widetilde{h_i}$ of those heights need to have an additional 1 added to them to be the true heights. We select the rows for which $Hg_i \bmod G$ is maximal. If there are two rows with the same remainder we prefer the first and last rows. Finally, if there is still a tie we work from the top row down.

References

Abbe, E. (1873). Beitrage zur Theorie des Mikroskops und der mikroskopischen Wahrmehmung. *Archiv für mikroskopische Anatomie, 9*, 413–420. doi:10.1007/BF02956173

Abbott, A. (2013, July 17). Solving the brain. *Nature, 499*, 272-274. doi:10.1038/499272a

Abbott, L. F. (1999). Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain Research Bulletin, 50*(5/6), 303-304. doi:10.1016/S0361-9230(99)00161-6

Acker, C. D., & White, J. A. (2007, October). Roles of IA and morphology in action potential propagation in CA1 pyramidal cell dendrites. *Journal of Computational Neuroscience, 23*(2), 201-216. doi:10.1007/s10827-007-0028-8

Adams, S. R., Mackey, M. R., Ramachandra, R., Lemieux, S. F., Steinbach, P., Bushong, E. A., Butko, M. T., Giepmans, B. N., Ellisman, M. H., & Tsien, R. Y. (2016, November 17). Multicolor Electron Microscopy for Simultaneous Visualization of Multiple Molecular Species. *Cell Chemical Biology, 23*(11), 1417-1427. doi:10.1016/j.chembiol.2016.10.006

Akita, T., & Kuba, K. (2000, November). Functional Triads Consisting of Ryanodine Receptors, Ca2+ Channels, and Ca2+-Activated K+ Channels in Bullfrog Sympathetic Neurons. *The Journal of General Physiology, 116*(5), 697–720.

Allen, B. A., & Levinthal, C. (1990, October). CARTOS II semi-automated nerve tracing: Three-dimensional reconstruction from serial section micrographs. *Computerized Medical Imaging and Graphics, 14*(5), 319-329. doi:10.1016/0895-6111(90)90106-L

Alliez, P., Jamin, C., Rineau, L., Tayeb, S., Tournois, J., & Yvinec, M. (2018). 3D Mesh Generation. In *CGAL User and Reference Manual* (4.12 ed.). CGAL Editorial Board. Retrieved from https://doc.cgal.org/4.12/Manual/packages.html#PkgMesh_3Summary

Almog, M., & Korngreen, A. (2016, August). Is realistic neuronal modeling realistic? *Journal of Neurophysiology, 116*, 2180-2209. doi:0.1152/jn.00360.2016

Alonso, G., & Widmer, H. (1997, April). Clustering of KV4.2 potassium channels in postsynaptic membrane of rat supraoptic neurons: an ultrastructural study. *Neuroscience, 77*(3), 617-621.

An, W. F., Bowlby, M. R., Betty, M., Cao, J., Ling, H. P., Mendoza, G., Hinson, J. W., Mattsson, K. I., Strassle, B. W., Trimmer, J. S., & Rhodes, K. J. (2000, February 3). Modulation of A-type potassium channels by a family of calcium sensors. *Nature, 403*(6769), 553-556. doi:10.1038/35000592

Anderson, J. R., Mohamed, S., Grimm, B., Jones, B. W., Koshevoy, P., Tasdizen, T., Whitaker, R., & Marc, R. E. (2010, December 10). The Viking viewer for connectomics: Scalable multi-user annotation and summarization of large volume data sets. *The Journal of Microscopy, 241*(1), 13-28. doi:10.1111/j.1365-2818.2010.03402.x

Anwar, H., Roome, C. J., Nedelescu, H., Chen, W., Kuhn, B., & De Schutter, E. (2014, July 23). Dendritic diameters affect the spatial variability of intracellular calcium dynamics in computer models. *Frontiers in Cellular Neuroscience, 168*, 8. doi:doi.org/10.3389/fncel.2014.00168

Araya, R., Vogels, T. P., & Yuste, R. (2014, July 15). Activity-dependent dendritic spine neck changes are correlated with synaptic strength. *Proceedings of the National Academy of Sciences, 111*(28), E2895-E2904. doi:10.1073/pnas.1321869111

Arellano, J. I., Benavides-Piccione, R., DeFelipe, J., & Yuste, R. (2007, November). Ultrastructure of Dendritic Spines: Correlation Between Synaptic and Spine Morphologies. *Frontiers in Neuroscience, 1*(1), 131-143. doi:10.3389/neuro.01.1.1.010.2007

Arganda-Carreras, I., Seung, H. S., Cardona, A., & Schindelin, J. (2012). *ISBI 2012 Challenge: Segmentation of neuronal structures in EM stacks challenge*. Retrieved from http://brainiac2.mit.edu/isbi_challenge/

Arganda-Carreras, I., Seung, H. S., Vishwanathan, A., & Berger, D. (2013). *ISBI 2013 Challenge: 3D segmentation of neurites in EM images (SNEMI3D)*. Retrieved from http://brainiac2.mit.edu/SNEMI3D/

Arganda-Carreras, I., Turaga, S. C., Berger, D. R., Ciresan, D., Giusti, A., Gambardella, L. M., Schimdhuber, J., Laptev, D., Dwivedi, S., Buhmann, J. M., Liu, T., Seyedhosseini, M., Tasdizen, T., Kamentsky, L., Burget, R., Uher, V., Tan, X., Sun, C., Pham, T. D., Bas, E., Uzunbas, M. G., Cardona, A., Schindelin, J., & Seung, H. S. (2015). Crowdsourcing the creation of image segmentation algorithms for connectomics. *Frontiers in Neuroanatomy, 142*.

Ashhad, S., & Narayanan, R. (2013, April). Quantitative interactions between the A-type K+ current and inositol trisphosphate receptors regulate intraneuronal Ca2+ waves and synaptic plasticity. *The Journal of Physiology, 591*(7), 1645-1669. doi:10.1113/jphysiol.2012.245688

Azevedo, F. A., Carvalho, L. R., Grinberg, L. T., Farfel, J. M., Ferretti, R. E., Jacob Filho, W., Lent, R., & Herculano-Houzel, S. (2009, April 10). Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *The Journal of Comparative Neurology, 513*(5), 532-541. doi:10.1002/cne.21974

Bach, F., & Moulines, E. (2011). Non-asymptotic analysis of stochastic approximation algorithms for machine learning. *Advances in Neural Information Processing, 24*, 451-459.

Baggio, D. L. (2006). Intravascular Ultra Sound Image Segmentation. *Undergraduate Final Project – Technological Institute of Aeronautics, 75f*.

Bang, B. H., & Bang, F. B. (1957, December). Graphic reconstruction of the third dimension from serial electron microphotographs. *The Journal of Ultrastructure Research, 1*(2), 138-139. doi:10.1016/S0022-5320(57)80002-1

Banno, T., & Kohno, K. (1998, December 14). Conformational changes of the smooth endoplasmic reticulum are facilitated by L-glutamate and its receptors in rat Purkinje cells. *The Journal Comparative Neurology, 402*(2), 252-263. doi:10.1002/(SICI)1096-9861(19981214)402:2<252::AID-CNE9>3.0.CO;2-U

Beare, R., & Lehmann, G. (2006). The watershed transform in ITK-discussion and new developments. *The Insight Journal, 6*(1).

Beck, E. J., Bowlby, M., An, W. F., Rhodes, K. J., & Covarrubias, M. (2002, February). Remodelling inactivation gating of Kv4 channels by KChIP1, a small-molecular-weight calcium-binding protein. *The Journal of Physiology, 538*(Pt 3), 691-706.

Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S., & Smith, K. (2011, April). Cython: The Best of Both Worlds. *Computing in Science Engineering, 13*(2), 31-39. doi:10.1109/MCSE.2010.118

Belevich, I., Joensuu, M., Kumar, D., Vihinen, H., & Jokitalo, E. (2016, January 4). Microscopy Image Browser: A platform for segmentation and analysis of multidimensional datasets. *PLoS Biology, 14*(1), e1002340. doi:10.1371/journal.pbio.1002340

Bennett, M. D., & Leitch, I. J. (2005). Genome size evolution in plants. In T. R. Gregory, *The Evolution of the Genome* (pp. 89–162). San Diego: Elsevier.

Berengolts, A., & Lindenbaum, M. (2000, April 11-12). On the performance of connected components grouping. *The 21st IEEE Convention of the Electrical and Electronic Engineers*. doi:10.1109/EEEI.2000.924365

Berridge, M. J. (1998). Neuronal Calcium Signaling. *Neuron, 21*, 13-26.

Berrout, J., & Isokawa, M. (2009, July). Homeostatic and stimulus-induced coupling of the L-type Ca2+ channel to the ryanodine receptor in the hippocampal neuron in slices. *Cell Calcium, 46*(1), 30-38.

Betzig, B., Patterson, G. H., Sougrat, R., Lindwasser, O. W., Olenych, S., Bonifacino, J. S., Davidson, M. W., Lippincott, -S. J., & Hess, H. F. (2006). Imaging intracellular fluorescent proteins at nanometer resolution. *Science, 15*(313(5793)), 1642-1645.

Beucher, S., & Lantuéjoul, C. (1979, September 17-21). Use of Watersheds in Contour Detection. *International Workshop on Image Processing: Real-time Edge and Motion Detection/Estimation, 132*.

Bishop, H. I., Guan, D., Bocksteins, E., Parajuli, L. K., Murray, K. D., Cobb, M. M., Misonou, H., Zito, K., Foehring, R. C., & Trimmer, J. S. (2015, November 4). Distinct Cell- and Layer-Specific Expression Patterns and Independent Regulation of Kv2 Channel Subtypes in Cortical Pyramidal Neurons. *The Journal of Neuroscience, 35*(44), 14922-14942. doi:10.1523/JNEUROSCI.1897-15.2015

Bloodgood, B. L., & Sabatini, B. L. (2007, January 18). Nonlinear regulation of unitary synaptic signals by CaV(2.3) voltage-sensitive calcium channels located in dendritic spines. *Neuron, 53*(2), 249-260. doi:10.1016/j.neuron.2006.12.017

Bock, D. D., Lee, W.-C. A., Kerlin, A. M., Andermann, M. L., Hood, G., Wetzel, A. W., Yurgenson, S., Soucy, E. R., Kim, H. S., & Reid, R. C. (2011). Network anatomy and in vivo physiology of visual cortical neurons. *Nature, 471*(7337), 177-182. doi:10.1038/nature09802

Bond, C. T., Maylie, J., & Adelman, J. P. (1999, April). Small-conductance calcium-activated potassium channels. *Annals of the New York Academy of Science, 868*(1), 370-378. doi:10.1111/j.1749-6632.1999.tb11298.x

Bootman, M. D., Lipp, P., & Berridge, M. J. (2001, June). Review The organisation and functions of local Ca(2+) signals. *Journal of Cell Science, 114*(Pt 12), 2213-2222.

Bootman, M., Niggli, E., Berridge, M., & Lipp, P. (1997, March 1). Imaging the hierarchical Ca2+ signalling system in HeLa cells. *The Journal of Physiology, 499*((Pt 2)), 307-314.

Borenstein, E., Sharon, E., & Ullman, S. (2004, June). Combining Top-down and Bottom-up Segmentation. *Proceedings of IEEE Workshop on Perceptual Organization in Computer Vision, IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Botsch, M., & Kobbelt, L. (2004). A remeshing approach to multiresolution modeling. *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (pp. 185-192). ACM.

Botsch, M., Kobbelt, L., Pauly, M., Alliez, P., & Lévy, B. (2010). *Polygon mesh processing.* CRC press.

Bourne, J. N., & Harris, K. M. (2008). Balancing structure and function at hippocampal dendritic spines. *Annual Review of Neuroscience, 31*, 47-67. doi:10.1146/annurev.neuro.31.060407.125646

Bouwer, J. C., Deerinck, T. J., Bushong, E., Astakhov, V., Ramachandra, R., Peltier, S. T., & Ellisman, M. H. (2016, September). Deceleration of probe beam by stage bias potential improves resolution of serial block-face scanning electron microscopic images. *Advanced Structural and Chemical Imaging, 2*(11), 2198-0926. doi:10.1186/s40679-016-0025-y

Braitenberg, V. (2001). Brain size and number of neurons: an exercise in synthetic neuroanatomy. *Journal of Computational Neuroscience, 10*(1), 71–77.

Braverman, I. M., & Ken-Yen, A. (1983, December). Ultrastructure and three-dimensional reconstruction of several macular and papular telangiectases. *The Journal of Investigative Dermatology, 81*(6), 489-497. doi:10.1111/1523-1747.ep12555569

Braverman, M. S., & Braverman, I. M. (1986, March). Three-Dimensional Reconstructions of Objects from Serial Sections Using a Microcomputer Graphics System. *The Journal of Investigative Dermatology, 86*(3), 290-294. doi:10.1111/1523-1747.ep12285445

Briggman, K. L., & Denk, W. (2006, October). Towards neural circuit reconstruction with volume electron microscopy techniques. *Current Opinion in Neurobiology, 16*(5), 565-570. doi:10.1016/j.conb.2006.08.010

Briggman, K. L., Helmstaedter, M., & Denk, W. (2011, March 10). Wiring specific in the direction-selectivity of the retina. *Nature, 471*(7337), 183-188. doi:10.1038/nature09818

Broadwell, R. D., & Cataldo, A. M. (1983, September). The neuronal endoplasmic reticulum: its cytochemistry and contribution to the endomembrane system. I. Cell bodies and dendrites. *The Journal of Histochemistry and Cytochemistry, 31*(9), 1077-1088. doi:10.1177/31.9.6309951

Buckman, J. F., Hernández, H., Kress, G. J., Votyakova, T. V., Pal, S., & Reynolds, I. J. (2001, January 15). MitoTracker labeling in primary neuronal and astrocytic cultures: influence of mitochondrial membrane potential and oxidants. *Journal of Neuroscience Methods, 104*(2), 165-76. doi:10.1016/S0165-0270(00)00340-X

Bush, J., & Kiggins, J. (2010, March). Changing dendritic excitability with A-type K+ channels. *UCSD BGGN 260 Neurodynamics*.

Bygrave, F. L., & Benedetti, A. (1996, June). What is the concentration of calcium ions in the endoplasmic reticulum? *Cell Calcium, 19*(6), 547-551. doi:10.1016/S0143-4160(96)90064-0

Cacciola, A. F., Moeller, P., & Wein, R. (2018). CGAL and the Boost Graph Library. In *CGAL User and Reference Manual* (4.12 ed.). CGAL Editorial Board. Retrieved from https://doc.cgal.org/4.12/Manual/packages.html#PkgBGLSummary

Cacciola, F. (2018). Triangulated Surface Mesh Simplification. In *CGAL User and Reference Manual* (4.12 ed.). CGAL Editorial Board. Retrieved from https://doc.cgal.org/4.12/Manual/packages.html#PkgSurfaceMeshSimplificationSummary

Cai, X., Liang, C. W., Muralidharan, S., Kao, J. P., Tang, C.-M., & Thompson, S. M. (2004, October 14). Unique Roles of SK and Kv4.2 Potassium Channels in Dendritic Integration. *Neuron, 44*(2), 351-364. doi:10.1016/j.neuron.2004.09.026

Canny, J. (1986). A Computational Approach To Edge Detection. *IEEE Trans. Pattern Analysis and Machine Intelligence, 8*(6), 679–698.

Cappelli, R., Maio, D., & Maltoni, D. (2000). Combining fingerprint classifiers. *Proceedings of First International Workshop on Multiple Classifier Systems*, 351-361.

Cardona, A., Saalfeld, S., Preibisch, S., Schmid, B., Cheng, A., Pulokas, J., Tomancak, P., & Hartenstein, V. (2010). An Integrated Micro- and Macroarchitectural Analysis of the Drosophila Brain by Computer-Assisted Serial Section Electron Microscopy. *PLoS Biology, 8*(10), e1000502. doi:10.1371/journal.pbio.1000502

Cardona, A., Saalfeld, S., Schindelin, J., Arganda-Carreras, I., Preibisch, S., Longair, M., Tomancak, P., Hartenstein, V., & Douglas, R. J. (2012, June 19). TrakEM2 software for neural circuit reconstruction. *PLoS One, 7*(6), e38011. doi:10.1371/journal.pone.0038011

Cardona, A., Saalfeld, S., Schindelin, J., Arganda-Carreras, I., Preibisch, S., Longair, M., Tomancak, P., Hartenstein, V., & Douglas, R. J. (2012). TrakEM2 Software for Neural Circuit Reconstruction. *PLoS ONE, 7*(6), e38011.

Carnevale, N. T., & Hines, M. L. (2006). *The NEURON Book.* Cambridge, UK: Cambridge University Press. doi:10.1017/CBO9780511541612

Cerda, O., & Trimmer, J. S. (2010, December 10). Analysis and functional implications of phosphorylation of neuronal voltage-gated potassium channels. *Neuroscience Letters, 486*(2), 60–67. doi:10.1016/j.neulet.2010.06.064

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002, June). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research, 16*, 321-357.

Chen, B. L., Hall, D. H., & Chklovskii, D. B. (2006). Wiring optimization can relate neuronal structure and function. *Proceedings of the National Academy of Sciences, 103*(12), 4723-4728. doi:10.1016/j.conb.2010.08.002

Chen, X., Yuan, L. L., Zhao, C., Birnbaum, S. G., Frick, A., Jung, W. E., Schwarz, T. L., Sweatt, J. D., & Johnston, D. (2006, November 22). Deletion of Kv4.2 gene eliminates dendritic A-type K+ current and enhances induction of long-term potentiation in hippocampal CA1 pyramidal neurons. *The Journal of Neuroscience, 26*(47), 12143-12151. doi:10.1523/JNEUROSCI.2667-06.2006

Cheng, H., & Lederer, W. J. (2008, October). Calcium Sparks. *Physiological Reviews, 88*(4), 1491-545. doi:10.1152/physrev.00030.2007

Cheng, H., Lederer, W. J., & Cannell, M. B. (1993, October 29). Calcium sparks: elementary events underlying excitation-contraction coupling in heart muscle. *Science, 262*(5134), 740-744.

Chklovskii, D. B., Vitaladevuni, S., & Scheffer, L. K. (2010). Semi-automated reconstruction of neural circuits using electron microscopy. *Current Opinion in Neurobiology, 20*(5), 667-675. doi:10.1016/j.conb.2010.08.002

Chklovskii, D. B., Vitaladevuni, S., & Scheffer, L. K. (2010, October). Semi-automated reconstruction of neural circuits using electron microscopy. *Current Opinion in Neurobiology, 20*, 667-675. doi:10.1016/j.conb.2010.08.002

Chow, S. K., Hakozaki, H., Price, D. L., MacLean, N. A., Deerinck, T. J., Bouwer, J. C., Martone, M. E., Peltier, S. T., & Ellisman, M. H. (2006, May). Automated microscopy system for mosaic acquisition and processing. *Journal of Microscopy, 222*(2), 76-84. doi:10.1111/j.1365-2818.2006.01577.x

Churas, C., Perez, A. J., Hakozaki, H., Wong, W., Lee, D., Peltier, S. T., & Ellisman, M. H. (2017, October). Probability Map Viewer: near real-time probability map generator of serial block electron microscopy collections. *Bioinformatics, 33*(19), 3145–3147. doi:10.1093/bioinformatics/btx376

Clark, A., & et al. (2017). Pillow.

Coates, A., & Ng, A. Y. (2012). Learning Feature Representations with K-Means. In G. Montavon, G. B. Orr, & K. Müller, *Neural Networks: Tricks of the Trade* (Vol. 7700, pp. 561-580). Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-35289-8_30

Cohen, P. (2000, December). The regulation of protein function by multisite phosphorylation--a 25 year update. *Trends Biochemical Sciences, 25*(12), 596–601.

Collette, A. (2013). *Python and HDF5.* O'Reilly Media.

Coltuc, D., & Bolon, P. (1999, October). Strict ordering on discrete images and applications. *International Conference on Image Processing*, 24-28. doi:10.1109/ICIP.1999.817089

Coltuc, D., Bolon, P., & Chassery, J.-M. (2006, May). Exact histogram specification. *IEEE Transactions on Image Processing, 15*(5), 1143-1152. doi:10.1109/TIP.2005.864170

Coons, A. H., Creech, H. J., Norman, J. R., & Berliner, E. (1942, November). The Demonstration of Pneumococcal Antigen in Tissues by the Use of Fluorescent Antibody. *The Journal of Immunology, 45*(3), 159–170.

Cowan, W. M., & Wann, F. (1973, December). A computer system for the measurement of cell and nuclear sizes. *Journal of Microscopy, 99*(3), 331-348. doi:10.1111/j.1365-2818.1973.tb04630.x

Dalal, N., & Triggs, B. (2005, June). Histograms of Oriented Gradients for Human Detection. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. doi:10.1109/CVPR.2005.177

Daneels, D., Van Campenhout, D., Niblack, C. W., Equitz, W., Barber, R., & Fierens, F. (1993). Interactive Outlining: An improved approach using active contours. *SPIE Proceedings of Storage and Retrieval for Image and Video Databases*, 226-233.

Dani, A., Huang, B., Bergan, J., Dulac, C., & Zhuang, X. (2011). Superresolution imaging of chemical synapses in the brain. *Neuron, 68*, 843-856.

Davidovits, P., & Egger, M. D. (1969). Scanning laser microscope. *Nature, 223*(5208), 831. doi:10.1038/223831a0

de Nó, R. L. (1947). Participation of the core in electrotonic phenomena. *Studies from the Rockefeller Institute for Medical Research - Reprints, 132*, 1-34.

De Zeeuw, C. I., Hertzberg, E. L., & Mugnaini, E. (1995, February). The dendritic lamellar body: a new neuronal organelle putatively associated with dendrodendritic gap junctions. *The Journal of Neuroscience, 15*(2), 1587-1604.

Debunne, G. (2017, November 17). *libQGLViewer*, 2.7.1. Retrieved from http://libqglviewer.com

Deerinck, T. J., Bushong, E. A., Lev-Ram, V., Shu, X., Tsien, R. Y., & Ellisman, M. H. (2010). Enhancing serial block-face scanning electron microscopy to enable high resolution 3-D nanohistology of cells and tissues. *Microscopy and Microanalysis, 16*, 1138–1139.

Dejerine, J. J., & Dejerine-Klumpke, A. (1895). *Anatomie des centres nerveux* (Vol. 1). Paris: Rueff.

Denk, W., & Horstmann, H. (2004). Serial block-face scanning electron microscopy to reconstruct three-dimensional tissue nanostructure. *PLoS Biology, 2*(11), 1900-1909.

Denk, W., Strickler, J., & Webb, W. (1990). Two-photon laser scanning fluorescence microscopy. *Science, 248*(4951), 73–6. doi:10.1126/science.2321027

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik, 1*, 269–271.

Du, J., Haak, L. L., Phillips-Tansey, E., Russell, J. T., & McBain, C. J. (2000, January 1). Frequency-dependent regulation of rat hippocampal somato-dendritic excitability by the K+ channel subunit Kv2.1. *The Journal of Physiology. ;, 522*(Pt 1), 19-31.

Du, J., Tao-Cheng, J. H., Zerfas, P., & McBain, C. J. (1998). The K+ channel, Kv2.1, is apposed to astrocytic processes and is associated with inhibitory postsynaptic membranes in hippocampal and cortical principal neurons and inhibitory interneurons. *Neuroscience, 84*, 37-48.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research, 12*, 2121–2159.

Duménieu, M., Oulé, M., Kreutz, M. R., & Lopez-Rojas, J. (2017, April). The Segregated Expression of Voltage-Gated Potassium and Sodium Channels in Neuronal Membranes: Functional Implications and Regulatory Mechanisms. *Frontiers in Cellular Neuroscience, 11*(115), 115. doi:10.3389/fncel.2017.00115

Electrical Signals of Nerve Cells. (2001). In D. Purves, G. J. Augustine, D. Fitzpatrick, L. C. Katz, A.-S. LaMantia, J. O. McNamara, & S. M. Williams (Eds.), *Neuroscience* (2nd ed.). Sunderland, MA: Sinauer Associates. Retrieved from https://www.ncbi.nlm.nih.gov/books/NBK11111/

El-Hassar, L., Hagenston, A. M., D'Angelo, L. B., & Yeckel, M. F. (2011, July). Metabotropic glutamate receptors regulate hippocampal CA1 pyramidal neuron excitability via Ca^{2+} wave-dependent activation of SK and TRPC channels. *Journal of Physiology, 589*(Pt 13), 3211-3229. doi:10.1113/jphysiol.2011.209783

Evans, J. D. (2000, December). Analysis of a multiple equivalent cylinder model with generalized taper. *IMA Journal of Mathematics Applied in Medicine and Biology, 17*(4), 347-377.

Faber, E. S. (2009). Functions and modulation of neuronal SK channels. *Cell Biochemistry and Biophysics, 55*(3), 127-139. doi:10.1007/s12013-009-9062-7

Feldman, D. E. (2012, August 23). The spike timing dependence of plasticity. *Neuron, 75*(4), 556-571. doi:10.1016/j.neuron.2012.08.001

Feller, W. (1950). *An Introduction to Probability Theory and Its Applications* (2nd ed., Vol. 1). Wiley.

Ferrante, M., Migliore, M., & Ascoli, G. A. (2013, January 30). Functional Impact of Dendritic Branch-Point Morphology. *The Journal of Neuroscience, 33*(5), 2156-2165. doi:10.1523/JNEUROSCI.3495-12.2013

Fiala, J. C. (2005, April). The Journal of Microscopy. *Reconstruct: A free editor for serial section microscopy, 218*(1), 52-61. doi:10.1111/j.1365-2818.2005.01466.x

Field, D. J. (1987). Relations between the statistics of natural images and the response properties of cortical cells. *Journal of the Optical Society of America A, 4*(12), 2379-2394. doi:10.1364/JOSAA.4.002379

Fifková, E., Markham, J. A., & Delay, R. J. (1983). Calcium in the spine apparatus of dendritic spines in the dentate molecular layer. *Brain Research, 266*, 163-168. doi:10.1016/0006-8993(83)91322-7

FitzHugh, R. (1961, July). Impulses and physiological states in theoretical models of nerve membrane. *Biophysical Journal, 1*(6), 445–466.

Fitzpatrick, J. S., Hagenston, A. M., Hertle, D. N., Gipson, K. E., Bertetto-D'Angelo, L., & Yeckel, M. F. (2009, April 1). Inositol-1,4,5-trisphosphate receptor-mediated Ca2+ waves in pyramidal neuron dendrites propagate through hot spots and cold spots. *The Journal of Physiology, 587*(Pt 7), 1439-1459.

Fogel, I., & Sagi, D. (1989). Gabor filters as texture discriminator. *Biological Cybernetics, 61*(2). doi:10.1007/BF00204594

Foskett, J. K., White, C., Cheung, K. H., & Mak, D. O. (2007, April). Inositol trisphosphate receptor Ca2+ release channels. *Physiology Reviews, 87*(2), 593-658.

Fourcaud-Trocmé, N., Hansel, D., van Vreeswijk, C., & Brunel, N. (2003, December 17). How spike generation mechanisms determine the neuronal response to fluctuating inputs. *The Journal of Neuroscience, 23*(37), 11628-11640.

Fox, C. A., Rafols, J. A., & Cowan, W. M. (1975, January 15). Computer measurements of axis cylinder diameters of radial fibers and "comb" bundle fibers. *The Journal of Comparative Neurology, 159*(2), 201-223. doi:10.1002/cne.901590204

Fox, P. D., Haberkorn, C. J., Akin, E. J., Seel, P. J., Krapf, D., & Tamkun, M. M. (2015, June 1). Induction of stable ER–plasma-membrane junctions by Kv2.1 potassium channels. *Journal of Cell Science, 128*(11), 2096-2105. doi:10.1242/jcs.166009

Frady, E. P. (2009, March). Multicompartment model of synaptic plasticity. *UCSD BGGN 260 Neurodynamics*.

Frangi, A. F., Niessen, W. J., Vincken, K. L., & Viergever, M. A. (1998). Multiscale vessel enhancement filtering. *International Conference on Medical Image Computing and Computer-Assisted Intervention, 1496*, 130-137.

Fredman, M. L., & Tarjan, R. E. (1987, July). Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the Association for Computing Machinery, 34*(3), 596-615. doi:10.1145/28869.28874

Frigo, M., & Johnson, S. G. (2005). The Design and Implementation of FFTW3. *Proceedings of the IEEE, 93*(2), 216–231.

Furness, J. B., Kearney, K., Robbins, H. L., B, H., Selmer, I. S., Neylon, C. B., Chen, M. X., & Tjandra, J. J. (2004, May 31). Intermediate conductance potassium (IK) channels occur in human enteric neurons. *Autonomic Neuroscience, 112*(1-2), 93-97. doi:10.1016/j.autneu.2004.02.003

Gant, J. C., Chen, K. C., Norris, C. M., Kadish, I., Thibault, O., Blalock, E. M., Porter, N. M., & Landfield, P. W. (2011, February 2). Disrupting Function of FK506-Binding Protein 1b/12.6 Induces the Ca2+-Dysregulation Aging Phenotype in Hippocampal Neurons. *The Journal of Neuroscience, 31*(5), 1693-1703. doi:10.1523/JNEUROSCI.4805-10.2011

Gao, X., Loriot, S., & Tagliasacchi, A. (2018). Triangulated Surface Mesh Skeletonization. In *CGAL User and Reference Manual* (4.12 ed.). CGAL Editorial Board. Retrieved from https://doc.cgal.org/4.12/Manual/packages.html#PkgMeanCurvatureSkeleton3Summary

Garthwaite, G., Hajo, F., & Garthwaite, J. (1992, June). Morphological response of endoplasmic reticulum in cerebellar Purkinje cells to calcium deprivation. *Neuroscience, 48*(3), 681-688. doi:10.1016/0306-4522(92)90411-T

Gay, H., & Anderson, T. F. (1954, December 24). Serial sections for electron microscopy. *Science, 120*(3130), 1071-1073. doi:10.1126/science.120.3130.1071

Giepmans, B. N., Adams, S. R., Ellisman, M. H., & Tsien, R. Y. (2006). The fluorescent toolbox for assessing protein location and function. *Science, 312*, 217-224.

Giuly, R. J., Kim, K.-Y., & Ellisman, M. H. (2013). DP2: Distributed 3D Image Segmentation Using Micro-labor Workforce. *Bioinformatics, Oxford Journals*.

Golding, N. L., Kath, W. L., & Spruston, N. (2001, December). Dichotomy of Action-Potential Backpropagation in CA1 Pyramidal Neuron Dendrites. *Journal of Neurophysiology, 86*(6), 2998-3010. doi:10.1152/jn.2001.86.6.2998

Goldman, D. E. (1943). Potential, impedance, and rectification in membranes. *The Journal of General Physiology, 27*(1), 37-60.

Goldstein, S. S., & Rall, W. (1974, October). Changes of action potential shape and velocity for changing core conductor geometry. *Biophysical Journal, 14*(10), 731-757. doi:10.1016/S0006-3495(74)85947-3

Golgi, C. (1906, December 11). The neuron doctrine – theory and facts. *Nobel lecture*.

Gould, S., Fulton, R., & Koller, D. (2009). Decomposing a Scene into Geometric and Semantically Consistent Regions. *IEEE International Conference on Computer Vision*.

Gray, E. G. (1959). Electron microscopy of synaptic contacts on dendrite spines of the cerebral cortex. *Nature, 183*(4675), 1592-1593. doi:10.1038/1831592a0

Greene, M. J., Kim, J. S., & Seung, H. S. (2016). Analogous Convergence of Sustained and Transient Inputs in Parallel on and off Pathways for Retinal Motion Computation. *Cell Reports, 14*(8), 1892-1900. doi:10.1016/j.celrep.2016.02.001

Greenspan, P., Mayer, E. P., & Fowler, S. D. (1985). Nile Red, A Selective Fluorescent Stain for Intracellular Lipid Droplets. *Journal of Cell Biology, 100*(1), 965–973.

Gregory, T. R. (2005). Genome size evolution in animals. In T. R. Gregory, *The Evolution of the Genome* (pp. 3–87). San Diego: Elsevier.

Grynkiewicz, G., Poenie, M., & Tsien, R. Y. (1985). A new generation of Ca2+ indicators with greatly improved fluorescence properties. *The Journal of Biological Chemistry, 260*, 3440-3450.

Gulledge, A. T., & Stuart, G. J. (2005, November 2). Cholinergic inhibition of neocortical pyramidal neurons. *The Journal of Neuroscience, 25*(44), 10308-10320. doi:10.1523/JNEUROSCI.2697-05.2005

Gundersen, H. J., Bendtsen, T. F., Korbo, L., Marcussen, N., Moller, A., Nielsen, K., Nyengaard, J. R., Pakkenberg, B., Sorensen, F. B., Vesterby, A., & West, M. J. (1988, January). Some new simple and efficient stereological methods and their use in pathological research and diagnosis. *APMIS: Acta pathologica, microbiologica, et immunologica Scandinavica, 96*, 379-394. doi:10.1111/j.1699-0463.1988.tb05320.x

Guo, W., Malin, S. A., Johns, D. C., & Nerbonne, J. M. (2002, July). Modulation of Kv4-encoded K+ Currents in the Mammalian Myocardium by Neuronal Calcium Sensor-1. *Journal of Biological Chemistry, 277*, 26436-26443. doi:10.1074/jbc.M201431200

Gustafsson, M. G. (2000). Surpassing the lateral resolution limit by a factor of two using structured illumination microscopy. *The Journal of Microscopy, 195*, 10-16.

Hagenston, A. M., & Bading, H. (2011, November). Calcium signaling in synapse-to-nucleus communication. *Cold Spring Harbor Perspectives on Biology, 3*(11), a004564. doi:10.1101/cshperspect.a004564

Hagenston, A. M., Fitzpatrick, J. S., & Yeckel, M. F. (2008, February). MGluR-mediated calcium waves that invade the soma regulate firing in layer V medial prefrontal cortical pyramidal neurons. *Cerebral Cortex, 18*(2), 407-423. doi:10.1093/cercor/bhm075

Hagmann, P. (2005). From diffusion MRI to brain connectomics. *Thesis*. doi:10.5075/epfl-thesis-3230

Hampel, F. R. (1986). *Robust Statistics: The Approach Based on Influence Functions.* New York: Wiley.

Harris, K. M., & Stevens, J. K. (1988, December). Dendritic spines of rat cerebellar Purkinje cells: serial electron microscopy with reference to their biophysical characteristics. *The Journal of Neuroscience, 8*(12), 4455-4469.

Harris, K. M., & Stevens, J. K. (1989, August 1). Dendritic spines of CA 1 pyramidal cells in the rat hippocampus: serial electron microscopy with reference to their biophysical characteristics. *The Journal of Neuroscience, 9*(8), 2982-2997.

He, X., Zemel, R., & Carreira-Perpinan, M. (2004). Multiscale conditional random fields for image labeling. *Proceedings of Computer Vision and Pattern Recognition*, 695-703.

Herculano-Houzel, S., Mota, B., & Lent, R. (2006, August 8). Cellular scaling rules for rodent brains. *Proceedings of the National Academy of Sciences, 103*(32), 12138-12143. doi:10.1073/pnas.0604911103

Herrera, G. M., & Nelson, M. T. (2002). Differential regulation of SK and BK channels by Ca2+ signals from Ca2+ channels and ryanodine receptors in guinea-pig urinary bladder myocytes. *The Journal of Physiology, 541*(Pt 2), 483-492. doi:10.1113/jphysiol.2002.017707

Herrera, G. M., Heppner, T. J., & Nelson, M. T. (2000, July). Regulation of urinary bladder smooth muscle contractions by ryanodine receptors and BK and SK channels. *American Journal of Physiology: Regulatory, integrative and comparative physiology, 279*(1), R60-68. doi:10.1152/ajpregu.2000.279.1.R60

Hessler, D., Young, S. J., Carragher, B. O., Martone, M. E., Lamont, S., Whittaker, M., Milligan, R. A., Masliah, E., Hinshaw, J. E., & Ellisman, M. H. (1992, August). Programs for visualization in three-dimensional microscopy. *Neuroimage, 1*(1), 55-67. doi:10.1016/1053-8119(92)90007-A

Heymann, J. B., & Belnap, D. M. (2007, January). Bsoft: image processing and molecular modeling for electron microscopy. *The Journal of Structural Biology, 157*(1), 3-18. doi:10.1016/j.jsb.2006.06.006

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hodgkin, A. L., & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology, 117*(4), 500–544. doi:10.1113/jphysiol.1952.sp004764

Hodgkin, A. L., & Katz, B. (1949). The effect of sodium ions on the electrical activity of the giant axon of the squid. *The Journal of Physiology, 108*, 37-77.

Hodgkin, A. L., & Rushton, W. A. (1946, December 3). The electrical constants of a crustacean nerve fibre. *Proceedings of the Royal Society of Medicine, 134*(873), 444-479.

Hoffman, D. A., Magee, J. C., Colbert, C. M., & Johnston, D. (1997, June 26). K+ channel regulation of signal propagation in dendrites of hippocampal pyramidal neurons. *Nature, 387*(6636), 869-875. doi:10.1038/43119

Hoffmann, H. P., & Avers, C. J. (1973, August 24). Mitochondrion of yeast: ultrastructural evidence for one giant, branched organelle per cell. *Science, 181*(4101), 749-751.

Holbro, N., Grunditz, A., & Oertner, T. G. (2009, September). Differential distribution of endoplasmic reticulum controls metabotropic signaling and plasticity at hippocampal synapses. *Proceedings of the National Academy of Science, 106*(35), 15055-15060. doi:10.1073/pnas.0905110106

Hood, L., & Rowen, L. (2013). The Human Genome Project: big science transforms biology and medicine. *Genome Medicine, 5*(9), 79. doi:10.1186/gm483

Hoshi, T., Tian, Y., Xu, R., Heinemann, S. H., & Hou, S. (2013). Mechanism of the modulation of BK potassium channel complexes with different auxiliary subunit compositions by the omega-3 fatty acid DHA. *Proceedings of the National Academy of Sciences, 110*(12), 4822-4827.

Hummel, R. A. (1977). Image Enhancement by Histogram Transformation. *Computer Graphics and Image Processing, 6*, 184-195.

Iino, M., & Endo, M. (1992). Calcium-dependent immediate feedback control of inositol 1,4,5-trisphosphate-induced Ca2+ release. *Nature, 360*, 76–78.

Ikematsu, N., Dallas, M. L., Ross, F. A., Lewis, R. W., Rafferty, J. N., David, J. A., Suman, R., Peers, C., Hardie, D. G., & Evans, A. M. (2011, November). Phosphorylation of the voltage-gated potassium channel Kv2.1 by AMP-activated protein kinase regulates membrane excitability. *Proceeding of the National Academy of Sciences, 108*(44), 18132-18137. doi:10.1073/pnas.1106201108

Insel, T. R., Landis, S. C., & Collins, F. S. (2013). The NIH BRAIN Initiative. *Science, 340*, 687-688.

Intel Corporation. (2018). Intel Math Kernel Library.

Ito, T., Nuriya, M., & Yasui, M. (2010, April). Regulation of Kv2.1 phosphorylation in an animal model of anoxia. *Neurobiology of Disease, 83*(1), 85-91. doi:10.1016/j.nbd.2010.01.002

Izhikevich, E. M. (2003, November). Simple model of spiking neurons. *IEEE Transactions on Neural Networks, 14*(6), 1569-1572. doi:10.1109/TNN.2003.820440

Jain, R., Kasturi, R., & Schunck, B. G. (1995). *Machine Vision, Jain, Kasturi, Schunck, 1995, 168-169.* McGraw-Hill, Inc.

Jain, V., Murray, J. F., Roth, F., Turaga, S., Zhigulin, V., Briggman, K. L., Helmstaedter, M. N., Denk, W., & Seung, H. S. (2007). Supervised Learning of Image Restoration with Convolutional Networks. *2007 IEEE 11th International Conference on Computer Vision*, (pp. 1-8). Rio de Janeiro, Brazil. doi:10.1109/ICCV.2007.4408909

Japkowicz, N. (2000). The Class Imbalance Problem: Significance and Strategies. *Proceedings of the 2000 International Conference on Artificial Intelligence (IC-AI'2000): Special Track on Inductive Learning*.

Jasinska, M., Siucinska, E., Jasek, E., Litwin, J. A., Pyza, E., & Kossut, M. (2016). Effect of Associative Learning on Memory Spine Formation in Mouse Barrel Cortex. *Neural Plasticity*. doi:10.1155/2016/9828517

Jedlicka, P., & Vlachos, A. (2008). A role for the spine apparatus in LTP and spatial learning. *Behavioural Brain Research, 192*, 12-19. doi:10.1016/j.bbr.2008.02.033

Jedlicka, P., & Vlachos, A. (2008). A role for the spine apparatus in LTP and spatial learning. *Behavioural Brain Research, 192*, 12-19. doi:10.1016/j.bbr.2008.02.033

Jinno, S., Jeromin, A., & Kosaka, T. (2005). Postsynaptic and extrasynaptic localization of Kv4.2 channels in the mouse hippocampal region, with special reference to targeted clustering at gabaergic synapses. *Neuroscience, 134*(2), 483-494. doi:10.1016/j.neuroscience.2005.04.065

Johnston, D., & Narayanan, R. (2008, June). Active dendrites: colorful wings of the mysterious butterflies. *Trends in Neurosciences, 31*(6), 309-316. doi:10.1016/j.tins.2008.03.004

Jones, C., Liu, T., Ellisman, M., & Tasdizen, T. (2013, April). Semi-Automatic Neuron Segmentation in Electron Microscopy Images Via Sparse Labeling. *Proceedings of the 2013 IEEE 10th International Symposium on Biomedical Imaging*, 1304-1307.

Jones, S. L., To, M.-S., & Stuart, G. J. (2017, October 23). Dendritic small conductance calcium-activated potassium channels activated by action potentials suppress EPSPs and gate spike-timing dependent synaptic plasticity. *eLife, 6*, e30333. doi:10.7554/eLife.30333

Jorstad, A., Nigro, B., Cali, C., Wawrzyniak, M., Fua, P., & Knott, G. (2015). NeuroMorph: A Toolset for the Morphometric Analysis and Visualization of 3D Models Derived from Electron Microscopy Image Stacks. *Neuroinformatics, 13*, 83–92. doi:10.1007/s12021-014-9242-5

Kapuscinski, J. (1995). DAPI: A DNA-specific fluorescent probe. *Biotechnic & Histochemistry, 70*(5), 220-233.

Kass, M., Witkin, A., & Terzopoulos, D. (1987, June). Snakes: Active Contour Models. *Proceedings of the First International Conference on Computer Vision*, 259-268.

Kasthuri, N., Hayworth, K. J., Berger, D. R., Schalek, R. L., Conchello, J. A., Knowles-Barley, S., Lee, D., Vázquez-Reina, A., Kaynig, V., Jones, T. R., Roberts, M., Morgan, J. L., Tapia, J. C., Seung, H. S., Roncal, W. G., Vogelstein, J. T., Burns, R., Sussman, D. L., Priebe, C. E., Hanspeter, P., & Lichtman, J. W. (2015). Saturated reconstruction of a volume of neocortex. *Cell, 162*(3), 648-661. doi:10.1016/j.cell.2015.06.054

Kekre, H. B., & Gharge, S. M. (2010, July). Image Segmentation using Extended Edge Operator. *International Journal on Computer Science and Engineering, 2*(4), 1086-1091.

Kerr, R., Bartol, T. M., Kaminsky, B., Dittrich, M., Chang, J. C., Baden, S., Sejnowski, T. J., & Stiles, J. R. (2008, October 13). Fast Monte Carlo Simulation Methods for Biological Reaction-Diffusion Systems in Solution and on Surfaces. *SIAM Journal on Scientific Computing, 30*(6), 3126-3149. doi:10.1137/070692017

Kettner, L. (2018). 3D Polyhedral Surface. In *CGAL User and Reference Manual* (4.12 ed.). CGAL Editorial Board. Retrieved from https://doc.cgal.org/4.12/Manual/packages.html#PkgPolyhedronSummary

Kettner, L. (2018). Halfedge Data Structures. In *CGAL User and Reference Manual* (4.12 ed.). CGAL Editorial Board. Retrieved from https://doc.cgal.org/4.12/Manual/packages.html#PkgHDSSummary

Keys, R. G. (1981, December). Cubic Convolution Interpolation for Digital Image Processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing, ASSP-29*(6), 1153-1160.

Khronos Group. (2017, July 31). *OpenGL*, 4.6. Retrieved from https://opengl.org/

Kim, J. S., Greene, M. J., Zlateski, A., Lee, K., Richardson, M., Turaga, S. C., Purcaro, M., Balkam, M., Robinson, A., Behabadi, B. F., Campos, M., Denk, W., Seung, H. S., & the EyeWirers. (2014). Space-time wiring specificity supports direction selectivity in the retina. *Nature, 509*, 331-336.

Kim, J., Jung, S. C., Clemens, A. M., Petralia, R. S., & Hoffman, D. A. (2007, June 21). Regulation of dendritic excitability by activity-dependent trafficking of the A-type K+ channel subunit Kv4.2 in hippocampal neurons. *Neuron, 54*(6), 933-947. doi:10.1016/j.neuron.2007.05.026

Kim, S., Fernandez-Martinez, J., Nudelman, I., Shi, Y., Zhang, W., Raveh, B., Herricks, T., Slaughter, B., Hogan, J., Upla, P., Chemmama, I., Pellarin, R., Echeverria, I., Shivaraju, M., Chaudhury, A., Wang, J., Williams, R., Unruh, J., Greenberg, C., Jacobs, E., Yu, Z., Jason de la Cruz, M., Mironska, R., Stokes, D., Aitchison, J., Jarrold, M., Gerton, J., Ludtke, S., Akey, C., Chait, B., Sali, A., & Rout, M. (2018, March 22). Integrative structure and functional anatomy of a nuclear pore complex. *Nature, 555*, 475-482. doi:10.1038/nature26003

King, B., Rizwan, A. P., Asmara, H., Heath, N. C., Engbers, J. D., Dykstra, S., Bartoletti, T. M., Hameed, S., Zamponi, G. W., & Turner, R. W. (2015, April 14). IKCa Channels Are a Critical Determinant of the Slow AHP in CA1 Pyramidal Neurons. *Cell Reports, 11*(2), 175-182. doi:10.1016/j.celrep.2015.03.026

Kirizs, T., Kerti-Szigeti, K., Lorincz, A., & Nusser, Z. (2014, June). Distinct axo-somato-dendritic distributions of three potassium channels in CA1 hippocampal pyramidal cells. *The European Journal of Neuroscience, 39*(11), 1771-1783. doi:10.1111/ejn.12526

Kiselycznyk, C., Hoffman, D. A., & Holmes, A. (2012, March 2). Effects of genetic deletion of the Kv4.2 voltage-gated potassium channel on murine anxiety-, fear- and stress-related behaviors. *Biology of Mood and Anxiety Disorders, 2*(5). doi:10.1186/2045-5380-2-5

Klee, C. B., H, C. T., & Krinks, M. H. (1979, December). Calcineurin: a calcium- and calmodulin-binding protein of the nervous system. *Proceedings of the National Academy of Sciences, 76*(12), 6270-6273.

Knott, G., & Genoud, C. (2013, October 15). Is EM dead? *The Journal of Cell Science, 126*(20), 4545-4552. doi:10.1242/jcs.124123

Knuth, D. E. (1977). A Generalization of Dijkstra's Algorithm. *Information Processing Letters, 6*(1), 1-5. doi:10.1016/0020-0190(77)90002-3.

Koch, C., & Segev, I. (1999). *Methods in neuronal modeling: from ions to networks* (2nd ed.). Cambridge, Massachusetts: MIT Press.

Koizumi, S., Bootman, M. D., Bobanović, L. K., Schell, M. J., Berridge, M. J., & Lipp, P. (1999, January). Characterization of elementary Ca2+ release signals in NGF-differentiated PC12 cells and hippocampal neurons. *Neuron, 22*(1), 125-137.

Komendantov, A. O., & Ascoli, G. A. (2009, April). Dendritic excitability and neuronal morphology as determinants of synaptic efficacy. *Journal of Neurophysiology, 101*(4), 1847-1866. doi:10.1152/jn.01235.2007

Kremer, J. R., Mastronarde, D. N., & McIntosh, J. R. (1996, January). Computer Visualization of Three-Dimensional Image Data Using IMOD. *Journal of Structural Biology, 116*(1), 71-76. doi:10.1006/jsbi.1996.0013

Kremerskothen, J., Plaas, C., Kindler, S., Frotscher, M., & Barnekow, A. (2005, February). Synaptopodin, a molecule involved in the formation of the dendritic spine apparatus, is a dual actin/alpha-actinin binding protein. *Journal of Neurochemistry, 92*(3), 597-606.

Krichmar, J. L., Nasuto, S. J., Scorcioni, R., Washington, S. D., & Ascoli, G. A. (2002, June 21). Effects of dendritic morphology on CA3 pyramidal cell electrophysiology: a simulation study. *Brain Research, 941*(1-2), 11-28. doi:10.1016/S0006-8993(02)02488-5

Kubat, M., & Matwin, S. (1997). Addressing the Curse of Imbalanced Training Sets: One Sided Selection. (M. Kaufmann, Ed.) *Proceedings of the Fourteenth International Conference on Machine Learning*, 179–186.

Kuwajima, M., Spacek, J., & Harris, K. M. (2012). Beyond counts and shapes: Studying pathology of dendritic spines in the context of the surrounding neuropil through serial section electron microscopy. *Neuroscience, 251*, 75–89. doi:doi:10.1016/j.neuroscience.2012.04.061

Lang, W. (1968). Nomarski differential interference-contrast microscopy. *ZEISS Information, 70*, 114–120.

Larkum, M. E., Watanabe, S., Nakamura, T., Lasser-Ross, N., & Ross, W. N. (2003). Synaptically activated Ca2+ waves in layer 2/3 and layer 5 rat neocortical pyramidal neurons. *The Journal of Physiology, 549*, 471–488.

Le Beux, Y. J. (1972, July). Subsurface cisterns and lamellar bodies: Particular forms of the endoplasmic reticulum in the neurons. *Zeitschrift für Zellforschung und Mikroskopische Anatomie, 133*, 327-352. doi:10.1007/BF00307242

LeCun, Y., Bottou, L., Orr, G. B., & Müller, K.-R. (1998). Efficient Backprop. *Neural networks: Tricks of the trade*.

Lee, D. (2013). Scientific analysis by the crowd: A system for implicit collaboration between experts, algorithms, and novices in distributed work. *PhD Defense, University of California San Diego*.

Leighton, S. B. (1981, February). SEM images of block faces, cut by a miniature microtome within the SEM – A technical note. *Scanning Electron Microscopy, 2*, 73-76.

Levinthal, C., & Ware, R. (1972, March 31). Three Dimensional Reconstruction from Serial Sections. *Nature, 236*, 207-210. doi:10.1038/236207a0

Levitan, I. B. (1994). Modulation of ion channels by protein phosphorylation and dephosphorylation. *Annual Review of Physiology, 56*, 193-212. doi:10.1146/annurev.ph.56.030194.001205

Lewis, D., & Catlett, J. (1994). Heterogeneous Uncertainity Sampling for Supervised Learning. (M. Kaufmann, Ed.) *Proceedings of the Eleventh International Conference of Machine Learning*, 148-156.

Liao, P.-S., Chen, T.-S., & Chung, P.-C. (2001). A fast algorithm for multilevel thresholding. *JOURNAL OF INFORMATION SCIENCE AND ENGINEERING, 17*(5), 713-727.

Lim, S. T., Antonucci, D. E., Scannevin, R. H., & Trimmer, J. S. (2000, February). A novel targeting signal for proximal clustering of the Kv2.1 K+ channel in hippocampal neurons. *Neuron, 25*(2), 385-397. doi:10.1016/S0896-6273(00)80902-2

Ling, C., & Li, C. (1998). Data Mining for Direct Marketing Problems and Solutions. *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*.

Liu, C., Yuen, J., & Torralba, A. (2011). SIFT Flow: Dense Correspondence across Scenes and its Applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 33*(5), 978-994.

Liu, T., Jones, C., Seyedhosseini, M., & Tasdizen, T. (2014, April). A modular hierarchical approach to 3D electron microscopy image segmentation. *Journal of Neuroscience Methods, 226*(15), 88-102.

Liu, T., Jurrus, E., Seyedhosseini, M., & Tasdizen, T. (2012). Watershed merge tree classification for electron microscopy image segmentation. *ICPR*.

Liu, T., Seyedhosseini, M., & Tasdizen, T. (2016). Image segmentation using hierarchical merge tree. *IEEE Transactions on Image Processing, 25*, 4596-4607.

Liu, T., Zhang, M., Javanmardi, M., Ramesh, N., & Tasdizen. (2016). SSHMT: Semi-supervised hierarchical merge tree for electron microscopy image segmentation. *ECCV*.

Loop, C. (1987). Smooth Subdivision Surfaces Based on Triangles. *M.S. Mathematics Thesis*. University of Utah.

Loriot, S., Tournois, J., & Yaz, I. O. (2018). Polygon Mesh Processing. In *CGAL User and Reference Manual* (4.12 ed.). CGAL Editorial Board. Retrieved from https://doc.cgal.org/4.12/Manual/packages.html#PkgPolygonMeshProcessingSummary

Losonczy, A., Makara, J. K., & Magee, J. C. (2008, March 27). Compartmentalized dendritic plasticity and input feature storage in neurons. *Nature, 452*, 436-441. doi:doi:10.1038/nature06725

Lowe, D. G. (1999). Object recognition from local scale-invariant features. *Proceedings of the International Conference on Computer Vision, 2*, 1150-1157.

Lowe, D. G. (2004, November). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision, 60*(2), 91-110. doi:10.1023/B:VISI.0000029664.99615.94

Lu, B., Zhang, Q., Wang, H., Wang, Y., Nakayama, M., & Ren, D. (2010, November 4). Extracellular Calcium Controls Background Current and Neuronal Excitability via an UNC79-UNC80-NALCN Cation Channel Complex. *Neuron, 68*(3), 488-499. doi:10.1016/j.neuron.2010.09.014

Lucchi, A., Smith, K., Achanta, R., Lepetit, V., & Fua, P. (2010). A Fully Automated Approach to Segmentation of Irregularly Shaped Cellular Structures in EM Images. (T. Jiang, N. Navab, J. P. Pluim, & M.

A. Viergever, Eds.) *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2010, 6362*, 463-471. doi:10.1007/978-3-642-15745-5_57

Lundstrom, B. N., Higgs, M. H., Spain, W. J., & Fairhall, A. L. (2008, November). Fractional differentiation by neocortical pyramidal neurons. *Nature Neuroscience, 11*(11), 1335-1342. doi:10.1038/nn.2212

Luo, D., Yang, D., Lan, X., Li, K., Li, X., Chen, J., Zhang, Y., Xiao, R. P., Han, Q., & Cheng, H. (2008, February). Nuclear Ca2+ sparks and waves mediated by inositol 1,4,5-trisphosphate receptors in neonatal rat cardiomyocytes. *Cell Calcium, 43*(2), 165-174.

Macagno, E. R., Levinthal, C., & Sobel, I. (1979). Three-dimensional computer reconstruction of neurons and neuronal assemblies. *Annual Reviews of Biophysics and Bioengineering, 8*, 323-351. doi:10.1146/annurev.bb.08.060179.001543

Magee, J. C., & Johnston, D. (1997, January 10). A synaptically controlled, associative signal for Hebbian plasticity in hippocampal neurons. *Science, 275*(5297), 209-213. doi:10.1126/science.275.5297.209

Mainen, Z. F., & Sejnowski, T. J. (1996, July 25). Influence of dendritic structure on firing pattern in model neocortical neurons. *Nature, 382*(6589), 363-366. doi:10.1038/382363a0

Makara, J. K., Losonczy, A., Wen, Q., & Magee, J. C. (2009, December). Experience-dependent compartmentalized dendritic plasticity in rat hippocampal CA1 pyramidal neurons. *Nature Neuroscience, 12*(12), 1485-1487. doi:10.1038/nn.2428

Mandikian, D. N., Cerda, O., Evans, A., Schneider, D., & Trimmer, J. S. (2012). Association and colocalization of plasma membrane Kv2.1 voltage-gated potassium channels with intracellular ryanodine receptor calcium-release channels. *Neuroscience 2012 Abstracts*, Program No. 330.04/C36.

Mandikian, D., Bocksteins, E., Parajuli, L. K., Bishop, H. I., Cerda, O., Shigemoto, R., & Trimmer, J. S. (2014, July 14). Cell Type Specific Spatial and Functional Coupling Between Mammalian Brain Kv2.1 K+ Channels and Ryanodine Receptors. *The Journal of Comparative Neurology, 522*(15), 3555-3574. doi:10.1002/cne.23641

Manita, S., & Ross, W. N. (2010, April). IP3 mobilization and diffusion determine the timing window of Ca2+ release by synaptic stimulation and a spike in rat CA1 pyramidal cells. *Hippocampus, 20*(4), 524-539. doi:10.1002/hipo.20644

Maravall, M., Koh, I. Y., Lindquist, W. B., & Svoboda, K. (2004, June). Experience-dependent changes in basal dendritic branching of layer 2/3 pyramidal neurons during a critical period for developmental plasticity in rat barrel cortex. *Cerebral Cortex, 14*(6), 655-664. doi:10.1093/cercor/bhh026

Maravall, M., Mainen, Z. F., Sabatini, B. L., & Svoboda, K. (2000, May). Estimating intracellular calcium concentrations and buffering without wavelength ratioing. *Biophysical Journal, 78*(5), 2655-2667.

Marrion, N. V., & Tavalin, S. J. (1998, October 29). Selective activation of Ca2+-activated K+ channels by co-localized Ca2+ channels in hippocampal neurons. *Nature, 395*, 900-905. doi:doi:10.1038/27674

Martin, D., Fowlkes, C., & Malik, J. (2002, December). Learning to Detect Natural Image Boundaries Using Brightness and Texture. *NIPS*.

Martone, M. E., Zhang, Y., Simpliciano, V. M., Carragher, B. O., & Ellisman, M. H. (1993, November). Three-dimensional visualization of the smooth endoplasmic reticulum in Purkinje cell dendrites. *Journal of Neuroscience, 13*(11), 4636-4646.

Marx, V. (2013). Neuroscience waves to the crowd. *Nature Methods, 10*, 1069-1074. doi:10.1038/nmeth.2695

Medeiros, J. M., Böck, D., Weiss, G. L., Kooger, R., Wepf, R. A., & Pilhofera, M. (2018, July). Robust workflow and instrumentation for cryo-focused ion beam milling of samples for electron cryotomography. *Ultramicroscopy, 190*, 1-11. doi:10.1016/j.ultramic.2018.04.002

Merriam, L. A., Scornik, F. S., & Parsons, R. L. (1999, August). Ca(2+)-induced Ca(2+) release activates spontaneous miniature outward currents (SMOCs) in parasympathetic cardiac neurons. *Journal of Neurophysiology, 82*(2), 540-550. doi:10.1152/jn.1999.82.2.540

Migliore, M., Ferrante, M., & Ascoli, G. A. (2005). Signal Propagation in Oblique Dendrites of CA1 Pyramidal Cells. *Journal of neurophysiology, 94*(6), 4145-4155. doi:10.1152/jn.00521.2005

Migliore, M., Hoffman, D. A., Magee, J. C., & Johnston, D. (1999, July-August). Role of an A-Type K+ Conductance in the Back-Propagation of Action Potentials in the Dendrites of Hippocampal Pyramidal Neurons. *Journal of Computational Neuroscience, 7*(1), 5–15. doi:10.1023/A:1008906225285

Misonou, H., Menegola, M., Mohapatra, D. P., Guy, L. K., Park, K. S., & Trimmer, J. S. (2006, December 27). Bidirectional activity-dependent regulation of neuronal ion channel phosphorylation. *The Journal of Neuroscience, 26*(52), 13505-13514. doi:10.1523/JNEUROSCI.3970-06.2006

Misonou, H., Mohapatra, D. P., & Trimmer, J. S. (2005, October). Kv2.1: a voltage-gated K+ channel critical to dynamic control of neuronal excitability. *Neurotoxicology, 26*(5), 743-752. doi:10.1016/j.neuro.2005.02.003

Misonou, H., Mohapatra, D. P., Park, E. W., Leung, V., Zhen, D., Misonou, K., Anderson, A. E., & Trimmer, J. S. (2004, July). Regulation of ion channel localization and phosphorylation by neuronal activity. *Nature Neuroscience, 7*(7), 711-718. doi:10.1038/nn1260

Miyazaki, K., & Ross, W. N. (2013, November 6). Ca2+ sparks and puffs are generated and interact in rat hippocampal CA1 pyramidal neuron dendrites. *The Journal of Neuroscience, 33*(45), 17777-17788. doi:10.1523/JNEUROSCI.2735-13.2013

Miyazaki, K., Manita, S., & Ross, W. N. (2012). Developmental profile of localized spontaneous Ca2+ release events in the. *Cell Calcium, 52*(6), 422-432. doi:10.1016/j.ceca.2012.08.001

Moens, P. B., & Moens, T. (1981, May). Computer measurements and graphics of three-dimensional cellular ultrastructure. *The Journal of Ultrastructure Research, 75*(2), 131-141. doi:10.1016/S0022-5320(81)80129-3

Mohapatra, D. P., & Trimmer, J. S. (2006, January 11). The Kv2.1 C terminus can autonomously transfer Kv2.1-like phosphorylation-dependent localization, voltage-dependent gating, and muscarinic modulation to diverse Kv channels. *The Journal of Neuroscience, 26*(2), 685-695. doi:10.1523/JNEUROSCI.4620-05.2006

Mohapatra, D. P., Misonou, H., Pan, S. J., Held, J. E., Surmeier, D. J., & Trimmer, J. S. (2009). Regulation of intrinsic excitability in hippocampal neurons by activity-dependent modulation of the KV2.1 potassium channel. *Channels, 3*(1), 46-56.

Morita, K. (2008, July 23). Possible role of dendritic compartmentalization in the spatial working memory circuit. *The Journal of Neurosci, 28*(30), 7699-7724. doi:10.1523/JNEUROSCI.0059-08.2008

Morris, C., & Lecar, H. (1981, July). Voltage Oscillations in the barnacle giant muscle fiber. *Biophysics Journal, 35*(1), 193-213. doi:10.1016/S0006-3495(81)84782-0

Mortensen, E., & Barrett, W. (1995). Intelligent Scissor for Image Composition. *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, 191–198.

Mortensen, E., Morse, B., & Barrett, W. (1992, October). Adaptive Boundary Detection Using 'Live-Wire' Two-Dimensional Dynamic Programming. *IEEE Computers in Cardiology*, 635–638.

Movellan, J. R. (2008, September 9). *Tutorial on Gabor Filters.* Retrieved from UCSD Machine Perception Laboratory (MPLab): http://mplab.ucsd.edu/tutorials/gabor.pdf

Murakoshi, H., Shi, G., Scannevin, R. H., & Trimmer, J. S. (1997, November). Phosphorylation of the Kv2.1 K+ channel alters voltage-dependent activation. *Molecular Pharmacology, 52*(5), 821-828.

Nagumo, J., Arimoto, S., & Yoshizawa, S. (1962, October). An Active Pulse Transmission Line Simulating Nerve Axon. *Proceedings of the IRE, 50*(10), 2061-2070. doi:10.1109/JRPROC.1962.288235

Nakamura, T., Barbara, J. G., Nakamura, K., & Ross, W. N. (1999, November). Synergistic release of Ca2+ from IP3-sensitive stores evoked by synaptic activation of mGluRs paired with backpropagating action potentials. *Neuron, 24*(3), 727-737. doi:10.1016/S0896-6273(00)81125-3

Nakamura, T., Lasser-Ross, N., Nakamura, K., & Ross, W. N. (2002). Spatial segregation and interaction of calcium signalling mechanisms in rat hippocampal CA1 pyramidal neurons. *The Journal of Physiology, 543*, 465–480.

Nakamura, T., Nakamura, K., Lasser-Ross, N., Barbara, J. G., Sandler, V. M., & Ross, W. N. (2000, November 15). Inositol 1,4,5-trisphosphate (IP3)-mediated Ca2+ release evoked by metabotropic agonists and backpropagating action potentials in hippocampal CA1 pyramidal neurons. *The Journal of Neuroscience, 20*(22), 8365-8376.

National Institutes of Health. (2009, July 15). NIH Launches the Human Connectome Project to Unravel the Brain's Connections. Retrieved from http://www.nih.gov/news/health/jul2009/ninds-15.htm

Neymotin, S. A., McDougal, R. A., Sherif, M. A., Fall, C. P., Hines, M. L., & Lytton, W. W. (2015, April). Neuronal calcium wave propagation varies with changes in endoplasmic reticulum parameters: a computer model. *Neural Computation, 27*(4), 898-924. doi:10.1162/NECO_a_00712

Ngo-Anh, T. J., Bloodgood, B. L., Lin, M., Sabatini, B. L., Maylie, J., & Adelman, J. P. (2005, May). SK channels and NMDA receptors form a Ca2+-mediated feedback loop in dendritic spines. *Nature Neuroscience, 8*(5), 642-649. doi:10.1038/nn1449

Nikolova, M., & Steidl, G. (2014, December). Fast Ordering Algorithm for Exact Histogram Specification. *IEEE Transactions on Image Processing, 23*(12), 5274-5283. doi:10.1109/TIP.2014.2364119

Nikolova, M., Wen, Y.-W., & Raymond, C. (2013, July). Exact Histogram Specification for Digital Images Using a Variational Approach. *Journal of Mathematical Imaging and Vision, 46*(3), 309-325. doi:10.1007/s10851-012-0401-8

Nishimura, T., & Matsumoto, M. (1998, January). Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Trans. on Modeling and Computer Simulation, 8*(1), 3-30. doi:10.1145/272991.272995

Nolan, M. F., Malleret, G., Dudman, J. T., Buhl, D. L., Santoro, B., Gibbs, E., Vronskaya, S., Buzsáki, G., Siegelbaum, S. A., Kandel, E. R., & Morozov, A. (2004, November 24). A Behavioral Role for Dendritic Integration: HCN1 Channels Constrain Spatial Memory and Plasticity at Inputs to Distal Dendrites of CA1 Pyramidal Neurons. *Cell, 119*(5), 719-732. doi:10.1016/j.cell.2004.11.020

Nomarski, G. (1952, May 14). *US Patent No. 2924142.*

Noske, A. B., Costin, A. J., Morgan, G. P., & Marsh, B. J. (2008, Mar). Expedited approaches to whole cell electron tomography and organelle mark-up in situ in high-pressure frozen pancreatic islets. *The Journal of Structural Biology, 161*(3), 298-313. doi:10.1016/j.jsb.2007.09.015

Otsu, N. (1979, January). A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics, 9*(1), 62-66. doi:10.1109/TSMC.1979.4310076

Ouyang, K., Zheng, H., Qin, X., Zhang, C., Yang, D., Wang, X., Wu, C., Zhou, Z., & Cheng, H. (2005, August 23). Ca2+ sparks and secretion in dorsal root ganglion neurons. *Proceedings of the National Academy of Science USA, 102*(34), 12259-12264.

Ozawa, T. (2010). Modulation of ryanodine receptor Ca2+ channels (Review). *Molecular Medicine Reports, 3*(2), 199-204. doi:10.3892/mmr_00000240

Palade, G. E. (1952, November). The fine structure of mitochondria. *The Anatomical Record, 114*(3), 427-451. doi:10.1002/ar.1091140304

Palade, G. E., & Porter, K. R. (1954, December 1). Studies on the endoplasmic reticulum. I. Its identification in cells in situ. *The Journal of Experimental Medicine, 100*(6), 641-656. doi:10.1084/jem.100.6.641

Palay, S. L., & Palade, G. E. (1955, January 25). The fine structure of neurons. *The Journal of Biophysical and Biochemical Cytology, 1*(1), 69-88. doi:10.1083/jcb.1.1.69

Park, K. S., Mohapatra, D. P., Misonou, H., & Trimmer, J. S. (2006, August 18). Graded regulation of the Kv2.1 potassium channel by variable phosphorylation. *Science, 313*(5789), 976-979. doi:10.1126/science.1124254

Parker, I., & Ivorra, I. (1990, November 16). Localized all-or-none calcium liberation by inositol trisphosphate. *Science, 250*(4983), 977-979.

Peddie, C. J., & Collinson, L. M. (2014, June). Exploring the third dimension: Volume electron microscopy comes of age. *Micron, 61*, 9-19. doi:10.1016/j.micron.2014.01.009

Pedlar, C., & Tilly, R. (1966, December). A new method of serial reconstruction from electron micrographs. *Journal of the Royal Microscopial Society, 86*(2), 189-197. doi:10.1111/j.1365-2818.1966.tb05337.x

Peltola, M. A., Kuja-Panula, J., Lauri, S. E., Taira, T., & Rauvala, H. (2011, December). AMIGO is an auxiliary subunit of the Kv2.1 potassium channel. *EMBO Reports, 12*(12), 1293-1299. doi:10.1038/embor.2011.204

Peltola, M. A., Kuja-Panula, J., Liuhanen, J., Võikar, V., Piepponen, P., Hiekkalinna, T., Taira, T., Lauri, S. E., Suvisaari, J., Kulesskaya, N., Paunio, T., & Rauvala, H. (2016, January). AMIGO-Kv2.1 Potassium Channel Complex Is Associated With Schizophrenia-Related Phenotypes. *Schizophrenia Bulletin, 42*(1), 191-201. doi:10.1093/schbul/sbv105

Perez, A. J. (2014). The Automated Reconstruction and Analysis of High Resolution Spatial Models of Neuronal Microanatomy. *PhD Dissertation, University of California San Diego.*

Perez, A. J., Seyedhosseini, M., Deerinck, T. J., Bushong, E. A., Panda, S., Tasdizen, T., & Ellisman, M. H. (2014, November). A workflow for the automatic segmentation of organelles in electron microscopy image stacks. *Frontiers in Neuroanatomy, 8*(126). doi:10.3389/fnana.2014.00126

Perkins, G. A., Renken, C. W., Song, J. Y., Frey, T. G., Young, S. J., Lamont, S., Martone, M. E., Lindsey, S., & Ellisman, M. H. (1997, December). Electron Tomography of Large, Multicomponent Biological Structures. *The Journal of Structural Biology, 120*(3), 219-227. doi:10.1006/jsbi.1997.3920

Petersen, O. H., Tepikin, A., & Park, M. K. (2001, May). The endoplasmic reticulum: one continuous or several separate Ca(2+) stores? *Trends in Neurosciences, 24*(5), 271-276.

Pettersen, E. F., Goddard, T. D., Huang, C. C., Couch, G. S., Greenblatt, D. M., Meng, E. C., & Ferrin, T. E. (2004, October). UCSF Chimera - a visualization system for exploratory research and analysis. *The Journal of Computational Chemistry, 25*(13), 1605-1612. doi:10.1002/jcc.20084

Pichat, J., Iglesias, J. E., Yousry, T., Ourselin, S., & Modat, M. (2018, May). A Survey of Methods for 3D Histology Reconstruction. *Medical Image Analysis, 46*, 73-105. doi:10.1016/j.media.2018.02.004

Pielage, J., Cheng, L., Fetter, R. D., Carlton, P. M., Sedat, J. W., & Davis, G. W. (2008). A presynaptic giant ankyrin stabilizes the NMJ through regulation of presynaptic microtubules and transsynaptic cell adhesion. *Neuron, 58*, 195-209.

Plaza, S. M., Scheffer, L. K., & Saunders, M. (2012, September 21). Minimizing manual image segmentation turn-around time for neuronal reconstruction by embracing uncertainty. *PLoS One, 7*(9), e44448. doi:10.1371/journal.pone.0044448

Poirazi, P., & Mel, B. W. (2001, March). Impact of active dendrites and structural plasticity on the memory capacity of neural tissue. *Neuron, 29*(3), 779-796. doi:10.1016/S0896-6273(01)00252-5

Poznanski, R. R. (Ed.). (1999). *Modeling in the Neurosciences: From Ionic Channels to Neural Networks* (1st ed.). Newark, NJ: CRC Press.

Poznanski, R. R., Cacha, L. A., Al-Wesabi, Y. M., Ali, J., Bahadoran, M., Yupapin, P. P., & Yunus, J. (2017, May 31). Solitonic conduction of electrotonic signals in neuronal branchlets with polarized microstructure. *Scientific Reports, 7*(1), 2746. doi:10.1038/s41598-017-01849-3

Prothero, J., & Prothero, J. (1982, December). Three-dimensional reconstruction from serial sections. I. A portable microcomputer-based software package in Fortran. *Computers and Biomedical Research, 15*(6), 598-604. doi:10.1016/0010-4809(82)90021-0

Qian, W. X., Yunquan, Z., & Yi, Q. (2013, November). AUGEM: Automatically Generate High Performance Dense Linear Algebra Kernels on x86 CPUs. *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'13)*.

Rall, W. (1959, November). Branching dendritic trees and motoneuron membrane resistivity. *Experimental Neurology, 1*(5), 491-527. doi:10.1016/0014-4886(59)90046-9

Rall, W. (1962). Electrophysiology of a Dendritic Neuron Model. *Biophysical Journal, 2*(2 Pt 2), 145-167.

Rall, W., Burke, R. E., Holmes, W. R., Jack, J. J., Redman, S. J., & Segev, I. (1992, October). Matching dendritic neuron models to experimental data. *Physiological Reviews, 72*(4 Suppl), S159-S186. doi:10.1152/physrev.1992.72.suppl_4.S159

Ramaswamy, S., & Markram, H. (2015, June 26). Anatomy and physiology of the thick-tufted layer 5 pyramidal neuron. *Frontiers in Cellular Neuroscience, 9*, 233. doi:10.3389/fncel.2015.00233

Ramón y Cajal, S. (1906, December 12). The structure and connexions of neurons. *Nobel Lecture*.

Rand, W. M. (1971). Objective Criteria for the Evaluation of Clustering Methods. *Journal of the American Statistical Association, 66*(336), 846-850. doi:10.1080/01621459.1971.10482356

Rigoglio, N. N., Silva, M. V., Junior, V. P., Lima, S. A., Nogueira, J. L., Fernandes, R. A., & Miglino, M. A. (2012, December). Ultrastructural morphometric analysis by transmission electron microscopy associated with stereology methods. (A. Mendez-Vilas, Ed.) *Current Microscopy Contributions to Advances in Science and Technology*, 309-315.

Rineau, L., & Yvinec, M. (2018). 3D Surface Mesh Generation. In *CGAL User and Reference Manual* (4.12 ed.). CGAL Editorial Board. Retrieved from https://doc.cgal.org/4.12/Manual/packages.html#PkgSurfaceMesher3Summary

Ritwik, K., Vázquez-Reina, A., & Pfister, H. (2010, June). Radon-like features and their application to connectomics. *IEEE Workshop on Mathematical Methods in Biomedical Image Analysis*, 186-193.

Roberto, S. J., García, R. B., & Wettstein, R. (1973, June 28). Serial sectioning study of some meiotic stages in Scaptericus borrelli (Grylloidea). *Chromosoma, 42*(3), 307-333. doi:10.1007/BF00284777

Rosenbluth, J. (1962). Subsurface cisterns and their relationship to the neuronal plasma membrane. *The Journal of Cell Biology, 13*, 405-421.

Ross, W. N. (2012, March). Understanding calcium waves and sparks in central neurons. *Nature Reviews Neuroscience, 13*(3), 157-168. doi:10.1038/nrn3168

Roy, J.-S. (2005). RandomKit: A library to generate random numbers.

Rudolph, S., & Thanawala, M. S. (2015, July). Location matters: somatic and dendritic SK channels answer to distinct calcium signals. *Journal of Neurophysiology, 114*(1), 1–5. doi:10.1152/jn.00181.2014

Rust, M. J., Bates, M., & Zhuang, X. (2006). Sub-diffraction-limit imaging by stochastic optical reconstruction microscopy (STORM). *Nature Methods, 3*, 793-795.

Sabatini, B. L., Oertner, T. G., & Svoboda, K. (2002). The life cycle of Ca2+ ions in dendritic spines. *Neuron, 33*, 439-452. doi:10.1016/s0896-6273(02)00573-1

Salah, Z., Orman, J., & Bartz, D. (2005). Live-Wire Revisited. *Bildverarbeitung für die Medizin 2005*, 158-162. doi:10.1007/3-540-26431-0_33

Schaefer, A. T., Larkum, M. E., Sakmann, B., & Roth, A. (2003, June). Coincidence detection in pyramidal neurons is tuned by their dendritic branching pattern. *Journal of Neurophysiology, 89*(6), 3143-3154. doi:10.1152/jn.00046.2003

Schalnat, G. E., Dilger, A., Bowler, J., & Randers-Pehrson, G. (2017, November 4). libpng. *1.6.34*. Retrieved from http://www.libpng.org/pub/png/libpng.html

Scharr, H. (2000, May). Optimal Operators in Digital Image Processing. *Dissertation*.

Schmutz, J., Wheeler, J., Grimwood, J., Dickson, M., Yang, J., Caoile, C., Bajorek, E., Black, S., Chan, Y. M., Denys, M., Escobar, J., Flowers, D., Fotopulos, D., Garcia, C., Gomez, M., Gonzales, E., Haydu, L., Lopez, F., Ramirez, L., Retterer, J., Rodriguez, A., Rogers, S., Salazar, A., Tsai, M., & Myes, R. M. (2004). Finishing the euchromatic sequence of the human genome. *Nature, 431*, 931-945. doi:10.1038/nature03001

Schrader, L. A., Birnbaum, S. G., Nadin, B. M., Ren, Y., Bui, D., Anderson, A. E., & Sweatt, J. D. (2006, March). ERK/MAPK regulates the Kv4.2 potassium channel by direct phosphorylation of the pore-forming subunit. *American Journal of Physiology: Cell Physiology, 290*(3), C852-C861. doi:10.1152/ajpcell.00358.2005

Schulien, A., Justice, J. A., Di Maio, R., Wills, Z. P., Shah, N. H., & Aizenman, E. (2016, May 15). Zn(2+) -induced Ca(2+) release via ryanodine receptors triggers calcineurin-dependent redistribution of cortical neuronal Kv2.1 K(+) channels. *The Journal of Physiology, 594*(10), 2647-2659. doi:10.1113/JP272117

Segal, M., & Korkotian, E. (2014, July 9). Endoplasmic reticulum calcium stores in dendritic spines. *Frontiers in Neuroanatomy, 8*, 64. doi:10.3389/fnana.2014.00064

Segal, M., & Vlachos, A. (2010). The Spine Apparatus, Synaptopodin, and Dendritic Plasticity. *The Neuroscientist, 16*, 125-131. doi:10.1177/1073858409355829

Senft, S. L. (2011, September). A Brief History of Neuronal Reconstruction. *Neuroinformatics, 9*(2–3), 119–128. doi:10.1007/s12021-011-9107-0

Seyedhosseini, M., & Tasdizen, T. (2015, August). Semantic Image Segmentation with Contextual Hierarchical Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 38*(5), 951-964.

Seyedhosseini, M., Kumar, R., Jurrus, E., Giuly, R., Ellisman, M., Pfister, H., & Tasdizen, T. (2011). Detection of neuron membranes in electron microscopy images using multi-scale context and radon-like features. *Med Image Comput Comput Assist Interv, 14*, 670-677.

Seyedhosseini, M., Sajjadi, M., & Tasdizen, T. (2013, December). Image segmentation with cascaded hierarchical models and logistic disjunctive normal networks. *Proceedings of the IEEE International Conference on Computer Vison (ICCV)*, 2168–2175. doi:10.1109/ICCV.2013.269

Shemer, I., Brinne, B., Tegnér, J., & Grillner, S. (2008, March 28). Electrotonic Signals along Intracellular Membranes May Interconnect Dendritic Spines and Nucleus. *PLoS Computational Biology, 4*(3), e1000036. doi:10.1371/journal.pcbi.1000036

Shi, G., Kleinklaus, A. K., Marrion, N. V., & Trimmer, J. S. (1994, September 16). Properties of Kv2.1 K+ channels expressed in transfected mammalian cells. *The Journal of Biological Chemistry, 269*(37), 23204-23211.

Shiue, L.-J. A. (2018). 3D Surface Subdivision Methods. In *CGAL User and Reference Manual* (4.12 ed.). CGAL Editorial Board. Retrieved from https://doc.cgal.org/4.12/Manual/packages.html#PkgSurfaceSubdivisionMethods3Summary

Siek, J., Lee, L.-Q., & Lumsdaine, A. (2001, December 20). The Boost Graph Library: User Guide and Reference Manual. Addison-Wesley Professional. Retrieved from http://www.boost.org/libs/graph/

Silver, R. A. (2010, July). Neuronal arithmetic. *Nature Reviews Neuroscience, 11*(7), 474–489. doi:10.1038/nrn2864

Simms, B. A., & Zamponi, G. W. (2014, April 2). Neuronal Voltage-Gated Calcium Channels: Structure, Function, and Dysfunction. *Neuron, 82*(1), 24-45. doi:10.1016/j.neuron.2014.03.016

Simpson, P. K. (1992). Fuzzy min-max neural networks. I. Classificationtion. *IEEE Transactions on Neural Networks, 3*(5), 776–786. doi:10.1109/72.159066

Sjöstrand, F. S. (1958, November). Ultrastructure of retinal rod synapses of the guinea pig eye as revealed by three-dimensional reconstructions from serial sections. *The Journal of Ultrastructure Research, 2*(1), 122-170. doi:10.1016/S0022-5320(58)90050-9

Sjöstrand, F. S. (1958, November). Ultrastructure of retinal rod synapses of the guinea pig eye as revealed by three-dimensional reconstructions from serial sections. *The Journal of Ultrastructure Research, 2*(1), 122-170. doi:10.1016/S0022-5320(58)90050-9

Sjöström, P. J., & Häusser, M. (2006, July 20). A cooperative switch determines the sign of synaptic plasticity in distal dendrites of neocortical pyramidal neurons. *Neuron, 51*(2), 227-238. doi:10.1016/j.neuron.2006.06.017

Sobel, I., & Feldman, G. (1968). A 3x3 Isotropic Gradient Operator for Image Processing. *Stanford Artificial Intelligence Project (SAIL)*.

Sommer, C., Straehle, C., Kothe, U., & Hamprecht, F. A. (2011, March-April). ilastik: Interactive learning and segmentation toolkit. *2011 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, 230-233. doi:10.1109/ISBI.2011.5872394

Sorensen, S. A., & Rubel, E. W. (2011, October). Relative Input Strength Rapidly Regulates Dendritic. *The Journal of Comparative Neurology, 519*(14), 2838-2851. doi:10.1002/cne.22656

Sorzano, C. O., Marabini, R., Velázquez-Muriel, J., Bilbao-Castro, J. R., Scheres, S. H., Carazo, J. M., & Pascual-Montano, A. (2004, November). XMIPP: a new generation of an open-source image processing package for electron microscopy. *The Journal of Structural Biology, 148*(2), 194-204. doi:10.1016/j.jsb.2004.06.006

Spacek, J., & Harris, K. M. (1997, January 1). Three-dimensional organization of smooth endoplasmic reticulum in hippocampal CA1 dendrites and dendritic spines of the immature and mature rat. *Journal of Neuroscience, 17*(1), 190-203.

Spangler, S. G. (1972, April). Expansion of the constant field equation to include both divalent and monovalent ions. *The Alabama Journal of Medical Sciences, 9*(2), 218-23.

Sporns, O., Tononi, G., & Kötter, R. (2005). The Human Connectome: A Structural Description of the Human Brain. *PLoS Computational Biology, 1*(4), e42. doi:10.1371/journal.pcbi.0010042

Spruston, N., Schiller, Y., Stuart, G., & Sakmann, B. (1995, April 14). Activity-dependent action potential invasion and calcium influx into hippocampal CA1 dendrites. *Science, 268*(5208), 297-300. doi:10.1126/science.7716524

Sterling, A. (2017, December 6). *5 Years of Eyewire*. Retrieved May 2017, from Blog | Eyewire: http://blog.eyewire.org/5-years-of-eyewire/

Stevens, B. J., & White, J. G. (1979). Computer reconstruction of mitochondria from yeast. *Methods in Enzymology, 56*, 718-728. doi:10.1016/0076-6879(79)56064-9

Stocker, M. (2004, October). Ca(2+)-activated K+ channels: molecular determinants and function of the SK family. *Nature Reviews Neuroscience, 5*(10), 758-770. doi:10.1038/nrn1516

Stuart, G. J., & Häusser, M. (2001, January). Dendritic coincidence detection of EPSPs and action potentials. *Nature Neuroscience, 4*(1), 63-71. doi:10.1038/82910

Stuart, G. J., & Sakmann, B. (1994, Jan 6). Active propagation of somatic action potentials into neocortical pyramidal cell dendrites. *Nature, 367*(6458), 69-72.

Stuart, G., Spruston, N., & Hausser, M. (Eds.). (2016). *Dendrites* (3rd ed.). Oxford, UK: Oxford University Press.

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. E. (2013, June). On the importance of initialization and momentum in deep learning. *In Proceedings of the 30th international conference on machine learning, 28*, 1139-1147.

Tagliasacchi, A., Alhashim, I., Olson, M., & Zhang, H. (2012, Auust). Mean curvature skeletons. *Computer Graphics Forum - Proceedings of the Symposium on Geometry Processing, 31*(5), 1735-1744. doi:10.1111/j.1467-8659.2012.03178.x

Tang, G., Peng, L., Baldwin, P. R., Mann, D. S., Jiang, W., Rees, I., & Ludtke, S. J. (2007). EMAN2: An extensible image processing suite for electron microscopy. *The Journal of Structural Biology, 157*(1), 38-46. doi:10.1016/j.jsb.2006.05.009

Tavanaei, A., & Maida, A. S. (2017). BP-STDP: Approximating Backpropagation using Spike Timing Dependent Plasticity. *Computing Research Repository*, abs/1711.04214.

Taylor, M. M. (1973). The Problem of Stimulus Structure in the Behavioural Theory of Perception. *South African Journal of Psychology, 3*, 23-45.

Teka, W., Marinov, T. M., & Santamaria, F. (2014, March 27). Neuronal Spike Timing Adaptation Described with a Fractional Leaky Integrate-and-Fire Model. *PLoS Computational Biology, 10*(3), e1003526. doi:10.1371/journal.pcbi.1003526

Terasaki, M., Slater, N. T., Fein, A., Schmidek, A., & Reese, T. S. (1994, August). Continuous network of endoplasmic reticulum in cerebellar Purkinje neurons. *Proceedings of the National Academy of Sciences, 91*(16), 7510-7514.

The CGAL Project. (2018). *CGAL User and Reference Manual* (4.12 ed.). CGAL Editorial Board. doi:https://doc.cgal.org/latest/Manual/index.html

The GIMP Help Team. (2015). *2.7 Inteligent Scissors*, 2.8. Retrieved September 2017, from GIMP Documentation: https://docs.gimp.org/en/gimp-tool-iscissors.html

The Qt Company. (2017, December 7). *Qt*, 5.10. Retrieved from https://www.qt.io/

Thomson, W. (1855). On the peristaltic induction of electric currents in submarine telegraph wires. *Mathematics and Physics Papers, 2*, 87.

Tian, L., Coghill, L. S., McClafferty, H., MacDonald, S. H., Antoni, F. A., Ruth, P., Knaus, H. G., & Shipston, M. J. (2004, August 10). Distinct stoichiometry of BKCa channel tetramer phosphorylation specifies channel activation and inhibition by cAMP-dependent protein kinase. *Proceedings of the National Academy of Sciences, 101*(32), 11897-11902. doi:10.1073/pnas.0402590101

Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*.

Tinati, R., Luczak-Roesch, M., Simperl, E., & Hall, W. (2017). An investigation of player motivations in Eyewire, a gamified citizen science project . *Computers in Human Behavior, 73*, 527-540. doi:10.1016/j.chb.2016.12.074

Titze, B., & Denk, W. (2013, May). Automated in-chamber specimen coating for serial blockface electron microscopy. *The Journal of Microscopy, 250*(2), 101-110. doi:10.1111/jmi.12023

Towns, J., Cockerill, T., Dahan, M., Foster, I., Gaither, K., Grimshaw, A., Hazlewood, V., Lathrop, S., Lifka, D., Peterson, G. D., Roskies, R., Scott, J. R., & Wilkins-Diehr, N. (2014, September-October). XSEDE: Accelerating Scientific Discovery. *Computing in Science & Engineering, 16*(5), 62-74. doi:10.1109/MCSE.2014.80

Tsien, R. Y. (1998). The green fluorescent protein. *Annual Review of Biochemistry, 67*, 509-544.

Tsubokawa, H., & Ross, W. N. (1996, November). IPSPs modulate spike backpropagation and associated [Ca2+]i changes in the dendrites of hippocampal CA1 pyramidal neurons. *Journal ofd Neurophysiology, 76*(5), 2896-2906. doi:10.1152/jn.1996.76.5.2896

Tustison, N. J., Yushkevich, P. A., & Gee, J. C. (2008, May). Live-Wire-ing the Insight Toolkit with Intelligent Scissors. *The Insight Journal*, http://hdl.handle.net/1926/1372.

Ujfalussy, B. B., Makara, J. K., Branco, T., & Lengyel, M. (2015). Dendritic nonlinearities are tuned for efficient spike-based computations in cortical circuits. *eLife, 4*, e10056. doi:10.7554/eLife.10056

Van, G. C. (1949). Applied hodology of the sympathetic trunk. *Confinia Neurologica, 9*(5-6), 344-365.

Vanhecke, D., Studer, D., & Ochs, M. (2007, September). Stereology meets electron tomography: Towards quantitative 3D electron microscopy. *Journal of Structural Biology, 159*(3), 443-450. doi:10.1016/j.jsb.2007.05.003

Varga, A. W., Yuan, L. L., Anderson, A. E., Schrader, L. A., Wu, G. Y., Gatchel, J. R., Johnston, D., & Sweatt, J. D. (2004, April 7). Calcium-calmodulin-dependent kinase II modulates Kv4.2 channel expression and upregulates neuronal A-type potassium currents. *The Journal of Neuroscience, 24*(14), 3643-3654. doi:10.1523/JNEUROSCI.0154-04.2004

Vergara, C., Latorre, R., Marrion, N. V., & Adelman, J. P. (1998, June). Calcium-activated potassium channels. *Current Opinion in Neurobiology, 8*(3), 321-329.

Vetter, P., Roth, A., & Häusser, M. (2001, February;). Propagation of action potentials in dendrites depends on dendritic morphology. *Journal of Neurophysiology, 85*(2), 926-937. doi:10.1152/jn.2001.85.2.926

Viola, P., & Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. *Computer Vision and Pattern Recognition*.

Walt, S. v., Colbert, Chris, S., & Varoquaux, G. (2011). The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering, 13*, 22-30. doi:10.1109/MCSE.2011.37

Walton, P. D., Airey, J. A., Sutko, J. L., Beck, C. F., Mignery, G. A., Südhof, T. C., Deerinck, T. J., & Ellisman, M. H. (1991, June). Ryanodine and inositol trisphosphate receptors coexist in avian cerebellar Purkinje neurons. *The Journal of Cell Biology, 113*(5), 1145-1157.

Wang, L., Dumoulin, A., Renner, M., Triller, A., & Specht, C. G. (2016, February 3). The Role of Synaptopodin in Membrane Protein Diffusion in the Dendritic Spine Neck. *PLOS One*. doi:10.1371/journal.pone.0148310

Wang, S. S.-H., Denk, W., & Häusser, M. (2000). Coincidence detection in single dendritic spines mediated by calcium release. *Nature Neuroscience, 3*, 1266-1273. doi:10.1038/81792

Wei, D. S., Mei, Y. A., Bagal, A., Kao, J. P., Thompson, S. M., & Tang, C. M. (2001, September 21). Compartmentalized and binary behavior of terminal dendrites in hippocampal pyramidal neurons. *Science, 293*(5538), 2272-2275. doi:10.1126/science.1061198

Wei, D., Jacobs, S., Modla, S., Zhang, S., Young, C. L., Cirino, R., Caplan, J., & Czymmek, K. (2012, July). High-resolution three-dimensional reconstruction of a whole yeast cell using focused-ion beam scanning electron microscopy. *BioTechniques, 53*(1), 41-48. doi:10.2144/000113850

White, J. G., & Amos, W. B. (1987). Confocal microscopy comes of age. *Nature, 328*, 183-184.

White, J. G., Southgate, E., Thomson, J. N., & Brenner, S. (1986, November 12). The structure of the nervous system of the nematode Caenorhabditis elegans. *Philosophical Transactions of the Royal Society B: Biological Sciences, 314*(1165), 1-340. doi:10.1098/rstb.1986.0056

Willey, T. J., Schultz, R. L., & Gott, A. H. (1973, July). Computer Graphics in Three Dimensions for Perspective Reconstruction of Brain Ultrastructure. *IEEE Transactions on Biomedical Engineering, BME-20*(4), 288-291. doi:10.1109/TBME.1973.324193

Wilmes, K. A., Schleimer, J. H., & Schreiber, S. (2017). Spike-timing dependent inhibitory plasticity to learn a selective gating of backpropagating action potentials. *The European Journal of Neuroscience, 45*(8), 1032-1043. doi:10.1111/ejn.13326

Wilt, B. A., Burns, L. D., Ho, E. T., Ghost, K. K., Mukamel, E. A., & Schnitzer, M. J. (2009). Advances in light microscopy for neuroscience. *Annual Review of Neuroscience, 32*, 435.

Wu, Y., Whiteus, C., Xue, C. S., Hayworthe, K. J., J, W., Hesse, H. F., & Camillia, P. D. (2017, June 13). Contacts between the endoplasmic reticulum and other membranes in neurons. *Proceedings of the National Academy of Sciences of the United States of America, 114*(24), E4859–E4867. doi:10.1073/pnas.1701078114

Yamada, S., Takechi, H., Kanchiku, I., Kita, T., & Kato, N. (2004, May). Small-conductance Ca2+-dependent K+ channels are the target of spike-induced Ca2+ release in a feedback regulation of pyramidal cell excitability. *Journal of Neurophysiology, 91*(5), 2322-2329. doi:10.1152/jn.01049.2003

Yamauchi, T. (2005, August). Neuronal Ca2+/Calmodulin-Dependent Protein Kinase II—Discovery, Progress in a Quarter of a Century, and Perspective: Implication for Learning and Memory. *Biological and Pharmaceutical Bulletin, 28*(8), 1342-1354. doi:10.1248/bpb.28.1342

Yao, J.-j., Zhao, Q.-R., Liu, D.-D., & Chow, C.-W. M.-A. (2016, June). Neuritin Upregulates Kv4.2 α-subunit of Potassium Channel Expression and Affects Neuronal. *Journal of Biological Chemistry, 291*, 17369-17381. doi:10.1074/jbc.M115.708883

Young, S. J., Royer, S. M., Groves, P. M., & Kinnamon, J. C. (1987, June). Three-dimensional reconstructions from serial micrographs using the IBM PC. *The Journal of Electron Microscopy Technique, 6*(2), 207-217. doi:10.1002/jemt.1060060211

Yuan, P., Leonetti, M. D., Pico, A. R., Hsiung, Y., & MacKinnon, R. (2010, July). Structure of the human BK channel Ca2+-activation apparatus at 3.0 A resolution. *Science, 329*(5988), 182-186. doi:10.1126/science.1190414

Zalk, R., Lehnart, S. E., & Marks, A. R. (2007). Modulation of the ryanodine receptor and intracellular calcium. *Annual Review of Biochemistry, 76*, 367-385. doi:10.1146/annurev.biochem.76.053105.094237

Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. *CoRR*.

Zhang, Y., & Huang, H. (2017, November). SK Channels Regulate Resting Properties and Signaling Reliability of a Developing Fast-Spiking Neuron. *The Journal of Neuroscience, 37*(44), 10738-10747. doi:10.1523/JNEUROSCI.1243-17.2017

Zhang, Y.-J. (2006). *Advances in Image and Video Segmentation.* Hershey, PA: IRM Press.

Zhao, X., Kuja-Panula, J., Sundvik, M., Chen, Y.-C., Aho, V., Peltola, M. A., Porkka-Heiskanen, T., Panula, P., & Rauvala, H. (2014). Amigo Adhesion Protein Regulates Development of Neural Circuits in Zebrafish Brain. *The Journal of Biological Chemistry, 289*, 19958-19975. doi:10.1074/jbc.M113.545582

Zuiderveld, K. (1994). Contrast Limited Adaptive Histograph Equalization. *Graphics Gems IV*, 474-485.