



An Interactive Deep Learning Toolkit for Automatic Segmentation of Images

Citation

Gonda, Felix E. 2016. An Interactive Deep Learning Toolkit for Automatic Segmentation of Images. Master's thesis, Harvard Extension School.

Link

<http://nrs.harvard.edu/urn-3:HUL.InstRepos:33797305>

Terms of use

This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Other Posted Material (LAA), as set forth at

<https://harvardwiki.atlassian.net/wiki/external/NGY5NDE4ZjgzNTc5NDQzMGIzZWZhMGFIOWI2M2EwYTg>

Accessibility

<https://accessibility.huit.harvard.edu/digital-accessibility-policy>

Share Your Story

The Harvard community has made this article openly available.
Please share how this access benefits you. [Submit a story](#)

An Interactive Deep Learning Toolkit for Automatic Segmentation of Images.

Felix E. Gonda

A Thesis in the Field of Software Engineering
for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

May 2016

Abstract

This thesis presents the design, analysis, and implementation of an interactive deep learning toolkit to reduce the manual labeling effort and improve the automatic segmentation of image data. While deep neural networks (DNN) models work well for image classification and segmentation, they require a large amount of training data, which is often acquired by manual annotations, a process that is time consuming and tedious. We develop our framework to train a deep neural network classifier in the background during the annotation process and provide the user with real-time feedback to guide the annotation effort. Our interactive system is developed in the context of segmentation of neuronal structures in Electron Microscopy (EM) images. This application is very important for the efficient mapping of brain structure and connectivity, enabling neuro-anatomists to gain new insight into the functional structure of the brain. The architecture of the system employs a three-tiered approach that provides a web-based user interface for distributed annotations, a learning model that trains a deep neural network classifier for segmentation purposes, and a web service middle-ware that facilitates data exchange between the user interface and the learning model.

Dedication

To my parents, Emmanuel Gonda and Tamara Poni,
who sacrificed so much for my upbringing and education.

To my brother Lawrence,
who instilled in me unbreakable faith in what is possible
and how to conduct myself as a leader and a deeply ethical person

To my brother Peter,
who sacrificed so much to shoulder the responsibilities of our family.

To my sisters Rosemary, Justina, Angelina, and Juliana,
who continues to inspire me daily.

To Jeremy Lund,
who instilled in me professional discipline and self-reliance.

To the Greving family: Brian, Katie, Alex, and Natalie
for their friendship and support.

To the Dilla family: Sam, Suzan, Kidden, Lowalia, Marsuk, and Moi
for their generosity and love through the years.

To my friends Steve Westhoff, Mary Gravitt, Jon Markee, and Hong Yuan
for their friendship and support.

Acknowledgments

First and foremost, I would like to thank my thesis advisor, Dr. Verena Kaynig-Fittkau, for infusing in me the passion for deep neural networks, computational neuroscience, and providing guidance in researching and conceptualizing this thesis project. Likewise, I extend thanks to my thesis director Dr. Jeff Parker for pushing me to refine my ideas and to express them at the highest level. I also would like to thank my academic advisor, Maura McGlame, for guiding me during the course of my degree. I am truly grateful to have been your student and thank you all for the great support and encouragement to pursue my own ideas

This project would not be possible without the resources and guidance of Prof. Hanspeter Pfister's Connectomics group. So I would like to extend my gratitude to Dr. Thouis Jones for providing access to resources and directions on troubleshooting, Daniel Haehn for access to Dojo and Mojo proofreading tools, and last but not least to Prof. Hanspeter Pfister from whom I learned computer graphics and visualization techniques.

Table of Contents

Table of Contents	vi
List of Figures	ix
List of Tables	xi
Chapter 1 Introduction	1
Motivation	2
Thesis Outline	7
Chapter 2 Background	8
Overview of Neural Anatomy	8
A Neuron	8
Connectome	10
Reconstructing a Connectome	10
Image Segmentation Techniques	12
Deep Neural Networks	14
Multilayer Perceptron (MLP)	18
Convolutional Neural Networks	22

Random Forests (RFs)	28
Data Collection and Curation	29
Summary	30
Chapter 3 Interactive Segmentation	32
Approach	32
System Architecture	33
Project	34
DNN Framework	38
Execution	39
Model Initialization	40
Sample Randomization	40
Reproducibility and Restartability	41
Hyper Parameters	42
Distributed Annotations	46
Web Service	52
Central Repository	53
Summary	54
Chapter 4 Results	55
Evaluation Methodology	55
Variation of Information Results	58
Strengths and Weaknesses	61
Summary	62
Chapter 5 Summary and Conclusions	63

Contributions	63
Future Work	65
References	70

List of Figures

Figure 1	Natural Segmentation	2
Figure 2	Membrane Segmentation	3
Figure 3	Interactive Annotation and Segmentation	5
Figure 4	Neuron Structure	9
Figure 5	Connectomics Reconstruction Pipeline	11
Figure 6	Artificial Neuron	15
Figure 7	Activation Functions	16
Figure 8	Multi-Layer Neural Network	18
Figure 9	CNN Weight Sharing	24
Figure 10	Convolution Layer	25
Figure 11	Convolutional Network	26
Figure 12	System Architecture	33
Figure 13	A Project	34
Figure 14	Project Editor	36
Figure 15	Training and Validation Sets	37
Figure 16	DNN Framework	38
Figure 17	Interactive Annotation Process	46

Figure 18 Image Browser	48
Figure 19 Annotation Editor	49
Figure 20 Segmentation Visualization	51
Figure 21 Web Service Architecture	52
Figure 22 Database Architecture	53
Figure 23 CNN Variation of Information Results	59
Figure 24 MLP Variation of Information Results	60

List of Tables

Table 1	Evaluation Configuration Settings	57
---------	---	----

Chapter 1 Introduction

“...the space systems community taught me a way of thinking that harnessed creative vision to physical, quantitative reasoning, in order to explore what could be achieved in new domains of engineering... Satellite launchers and moonships grew out of quantifiable engineering visions: system-level concepts that could be sketched, assessed, and discarded at a rapid pace, evolving through a kind of Darwinian competition. The best concepts would win the resources of time and attention needed to fill in more details, to optimize designs, to apply closer analysis, and after this refinement and testing, to compete again. The prize at the end would be a design refined into fully detailed specifications, then metal cut on a factory floor, then a pillar of fire rising into the sky bearing a vision made real... ”

– K. Eric Drexler

This thesis presents the design and implementation of an interactive method for automatic segmentation of images with the goal of reducing manual labeling effort, improving segmentation throughput, and minimizing errors in the process of classifying objects in images. It employs interactive techniques to assist the labeling of an image by simultaneously training a deep neural networks model in the background with the label data, while automatically segmenting the image in real-time. This method is applied here to the segmentation of neuronal structures in Electron Microscopy (EM) images, but its application extends to other

problem domains including facial recognition, video surveillance, traffic management, and many more. Its unifying aspiration is to develop a scalable technology architecture that can be applied to image segmentation problems in the Connectomics field of neuroscience and beyond.

Motivation

Image segmentation is the branch of computer vision whose purpose is to partition digital images into meaningful regions with the goal of locating objects and boundaries in images. At the fundamental level, the task of segmentation is finding the pixels that make up an object. Figure 1 depicts an example of image segmentation of a natural setting.

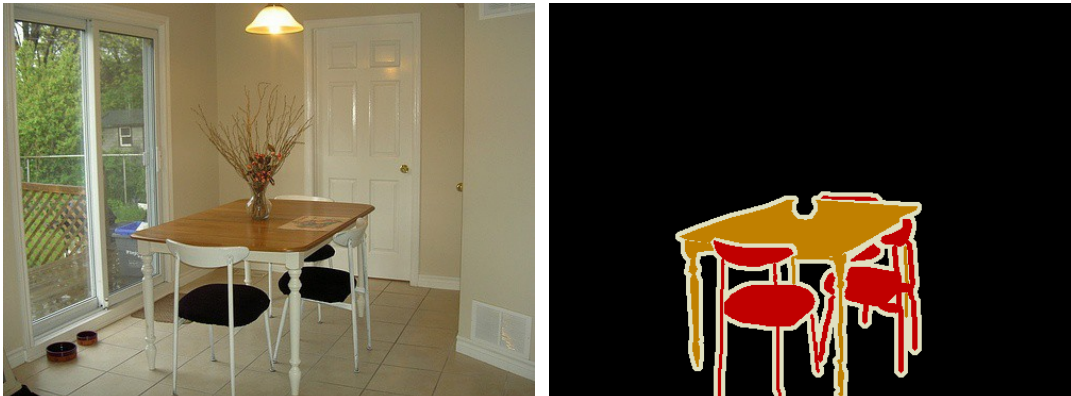


Figure 1: Example of natural image segmentation from PASCAL Visual Object Classes Challenge 2008 dataset. Left is original image and right is segmented image with objects grouped by color-coded labels. (Everingham et al., 2008)

The task of segmentation, typically, follows the process of grouping pixels (Picture Elements) with similar characteristics under labels. These groupings form the basis of detecting objects and boundaries in images. An image is said to be perfectly segmented if each

pixel is correctly assigned to the correct object. However, because of the acquisition methods of digital images, perfect image segmentation may not be possible since a pixel may straddle the boundaries of multiple objects. Most methods assign a pixel to one object, which may be adequate for many applications. However, because some pixels are ambiguous, or are on the boundary of objects, rather than assign them to a single object, they're assigned a probability distribution. This problem is further confounded by over-segmentation and under-segmentation, where in the former, pixels belonging to the same object are classified as belonging to different segments, and in the latter, pixels of different objects are classified as belonging to the same object.

This problem is no different in neuroscience, where the goal is to identify neuronal structures in brain images. Figure 2 depicts the segmentation of membranes in a brain image.

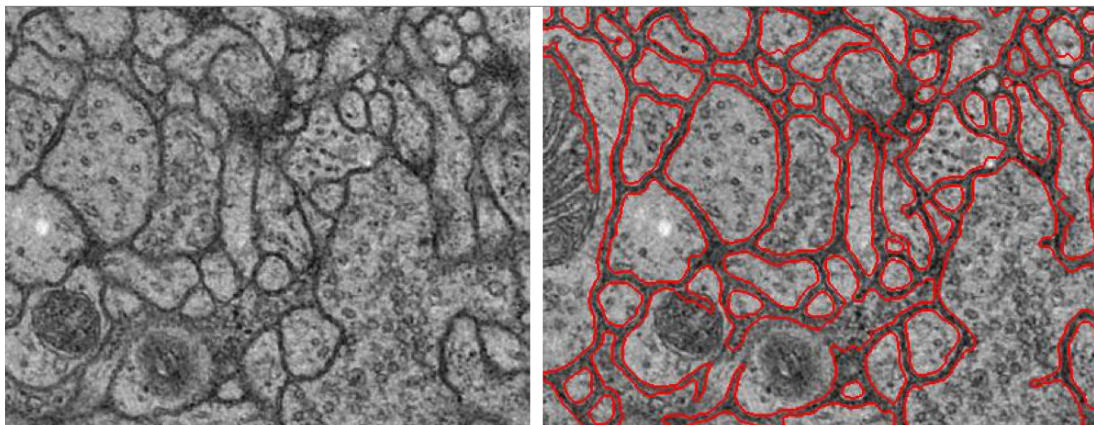


Figure 2: Example of membrane segmentation. Left is original image and right is segmented image. (Kaynig et al., 2010)

Many techniques have been applied to the problem of segmentation, ranging from manual to fully automated methods. Although manual segmentation remains the gold standard in many problem domains, it is impractical when the amount of data outpace the human ability

to analyze and segment due to the amount of time required. Furthermore, bias is introduced when comparing segmentation produced by different experts, which can affect the accurate statistical analysis of data. Therefore, the development of reliable and robust automatic segmentation methods is necessary.

In this thesis project, we focus on the central problem of automatic segmentation of neuronal structures by employing Deep Neural Networks (DNN) (Ciresan et al., 2011) (Schmidhuber, 2012) concepts to train a classifier for identifying structures in images of brain circuitry. DNNs are a class of machine learning algorithms whose architecture mimics the functions of the brain. A DNN architecture is composed of multiple non-linear processes that are organized into layers for the task of identifying patterns in data; a task that can be performed in supervised or non-supervised fashion. DNNs model hierarchical features in data by employing a system of inter-connected layers that can be activated by simulation of inputs and learning the system from data. The hierarchical nature of DNNs is suitable for image classification problems, and this thesis leverages these capabilities to perform automatic segmentation of neurons. Some of the key advantages of DNNs that compliment our goals include:

Flexibility

Whereas in traditional object detection algorithms the features are hand-crafted by computer vision experts, in supervised DNN architectures, the features are automatically learned by the network from data; and the human intervention is in supplying training data. This results into a general-purpose architecture that can be easily tailored to solve other problems without requiring an expert. In our application scenario, the training is interactively supervised by a human expert, and the infrastructure developed for detecting neuronal structures could be utilized to detect objects in videos, natural images, etc.

Scalability

The training process of DNN model scales very well with large data sets. It can go through million of images and make use of the input to learn features in the data. The more data you feed it, the smarter it gets. This suits the goals of Connectomics because of the large amount of brain data generated by Electron Microscopy.

Speed

A DNN model learns the connections between neurons in different layers, and these connections once learned, could be disconnected from the network and utilized with a fully-connected layer to generate predictions, thus improving performance. Furthermore, the parallel computational ability of neural networks makes them suitable for a real-time application such as the one developed in this thesis. As such, their training and execution can be improved when coupled with Graphic Processing Units (GPU) and multi-threaded Central Processing Units (CPU).

In our system, as depicted in Figure 3, the input data consists of images from Electron Microscopy (EM) and object class labels from user annotations. The labels are collected interactively from users in small windows centered around pixels forming a set of features.

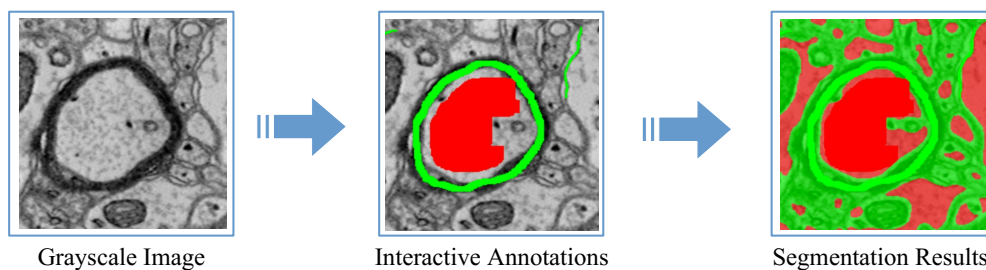


Figure 3: An illustration of the interactive labeling process using a DNN model.

Together, the input image and labels feed to a running DNN model which trains a classifier and partition images into neuronal structures. The results of segmentation produces a probability map that is visualized as an overlay to guide the user in the annotation process. This feedback loop occurs in real-time and enables quick training of a classifier for a segmentation task. A fully trained model can be used to process a large number of brain images automatically.

Our approach focus on the area of Connectomics. Connectomics is an emerging discipline of neuroscience that seeks efficient and high-throughput techniques for reconstructing neural connectivity data from brain images. In Connectomics, large volume of microscopy data of brain circuitry is acquired at nanometer-scale resolution; and from these images Connectomics aims to map the connectivity between neurons in the volume in support of efforts to reverse-engineer the brain. This mapping has direct application in neurology and psychiatry in that it could help us understand many diseases of the nervous system such as learning disorders in children and certain psychiatric diseases that are hard to diagnose such as Alzheimer, Autism, Anxiety disorders and many more (Kaynig et al., 2015) (Bergstra et al., 2010). It could also give us insight into how the brain stores and manages information from birth to old age. The challenge with Connectomics, however, is that the capacity to acquire the brain data far outpaces the ability to analyze it. A $1mm^3$ of rat cortex image alone is approximately 2 petabytes of data, and a complete rat cortex of $500mm^3$ equals 1000 petabytes, which would take many years to process and analyze. Therefore, tools and techniques that further the research and goals of Connectomics are extremely important to the neuroscience community.

Thesis Outline

In chapter 2, we give a background of the problem of identifying neuronal structures in brain images by reviewing neural anatomy and its relevant components to Connectomics. We review the concept of a connectome, which describes the synaptic connectivity between neurons in an organism's nervous system. We then discuss the state of the art methods in Connectomics that have been applied to the problem of segmenting neuronal structures, paying a special attention to Artificial Neural Networks. Then we conclude the chapter with a description of the nature and acquisition methods of the input data used in this project.

In Chapter 3, we present our approach to image segmentation by formulating an interactive framework for segmentation of images based on a deep neural network classifier trained on pixel annotations. We begin by giving a description of our approach followed by a detailed architecture of the software framework. We outline the process of annotating pixels, the process of training a classifier based on deep neural networks techniques, and the real-time feed-back loop by which annotations are transferred to the learning model and segmentation results are presented to the user.

Chapter 4 discusses the technical achievements of the project and report experimental results. We give a detail description of the evaluation methodology employed, the measurement metrics used, and the training data set. We then present results of evaluating an MLP and a CNN model in interactive and offline modes.

Chapter 5 summarizes the contributions made by this thesis and discusses directions for future research.

Chapter 2 Background

In this chapter we give an overview of neural anatomy and how the problem of identifying neuronal structures in brain images relates to the overall goal of reconstructing the brain. We then explore the state of the art methods in Connectomics currently used in image segmentation, highlighting their strengths and weaknesses, and their application scenarios. Then we conclude the chapter with a description of the input data used in this thesis.

Overview of Neural Anatomy

The Neuroscience field of Connectomics brings together a variety of existing approaches for understanding the functional and structural connectivity of the brain. This effort involves the acquisition of large volumes of images of brain circuitry in order to map the brain at a cellular level and the development of tools to analyze the data. To better understand the goal of Connectomics, we give an overview of neural anatomy and its relevant components to Connectomics.

A Neuron

The brain is the processing center of the nervous system. Its major function is to interpret electrical signals from the body's nerve cells, control its functions, and keep it alive. Inside the brain is a network of over 100 billion nerve cells, called neurons, that produces chemicals called neurotransmitters which transmit signals across neurons and from neurons to

other cells (Freeley and Pengelley, 2010). This network is responsible for our five senses and controls our motion and consciousness. A neuron, as depicted in Figure 4, is composed of four essential parts with differing functions: the cell body, dendrites, axon, and axon terminals.

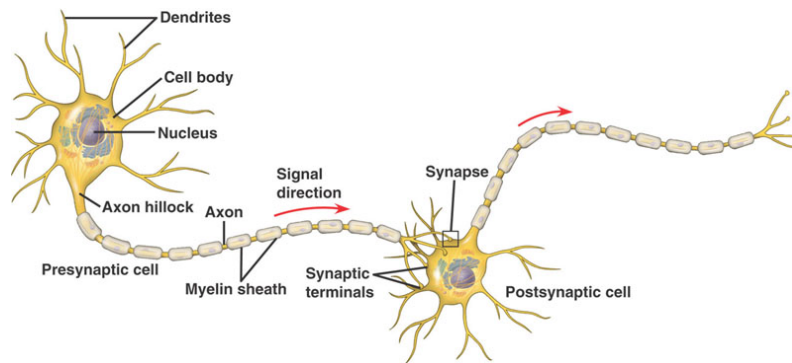


Figure 4: An illustration of two neurons connected by a synapse, figure adapted from Dirnberger (2016)

The cell body holds the nucleus and most of the organelles in the cell including the dendrites and axons. Most neurons have a single axon that extends outward from the cell body toward an axon terminus, forming presynaptic terminals that relay information to dendrites. The axon terminals form connections with other cells and provide the means by which neurotransmitter molecules are exchanged. A neuron communicates with other neurons by generating electrical signals that activate synapses and emit neurotransmitter molecules. Emitted molecules travel through postsynaptic terminals and are sensed by receptor molecules. This transmission process provides direct connectivity between cells and is fundamental in facilitating the operations of the brain (Kandel et al., 2013) (Seung, 2012). Each individual neuron can be connected to hundreds or thousands of other cells in networks that can extend over large volumes that constitute a connectome of brain circuitry.

Connectome

A connectome is the wiring diagram of all the neurons in an animal, and the goal of Connectomics is to create connectomes of brains. Our connectome consists of all the neurons in our body, their structural connections and functional interactions. So far, the only organism for which a connectome exists is a version of the small nematode worm *Caenorhabditis elegans* (*C. elegans*). The nervous system of the *C. elegans* consist of a network of about three hundred neurons and seven thousand synaptic connections, which were fully mapped in the 1970s and 1980s (White et al., 1986). The human connectome, however, is much more complicated than this. It has an estimated 100 billion neurons, and 10 thousand times that many connections. The information contained in the connectome is unknown, but neuroscientists believe that our memories and the information that makes us who we are is encoded in a connectome (Seung, 2012). They believe that reconstructing these neural synaptic connectivity would enable us to better understand diseases that manifest as mental disorders and aid biologist in finding solutions (Morgan and Lichtman, 2010). To test these hypotheses we need advanced technologies to find and reconstruct connectomes.

Reconstructing a Connectome

The scale of neuron connectivity in a connectome requires the capture of large volumes of tissue data in order to reconstruct the neuron wiring at a cellular level. This effort requires both hardware and software technologies to capture and analyze brain circuitry data. Several methods exist in the Neuroscience community for imaging brain data, chief among them are Magnetic Resonance Imaging (MRI) and Electron Microscopy. Many Connectomics techniques utilize EM methods because their shorter wavelengths of high-energy electrons is able to capture a much higher lateral resolution at the synaptic level (Briggman

and Bock, 2012). These imaging techniques allow us to acquire brain data more rapidly and the bottleneck now lies in the accurate reconstruction of neurons from the volume images. One approach to reconstruction is to manually trace the contours of a neuron on a slice and link consecutive slices together to create a 3D topology of the circuitry (Mishchenko et al., 2010). This approach, however, is not without problems. Human errors, such as mistracing due to attention-related mistakes or when different interpretation of difficult areas of data cause variation in tracings can lead to miss-assignment of synapses to neurons (Helmstaedter et al., 2011). But more importantly, humans are slow at tracing, and the rate at which data is acquired outpaces a human ability to trace by hand. As such, semi and fully automated approaches are required in order to bridge the gap between acquisition and analysis.

The Connectomics group at Harvard John A. Paulson School of Engineering and Applied Sciences developed a pipeline, depicted in Figure 5, for reconstructing the brain circuitry of a mammal.

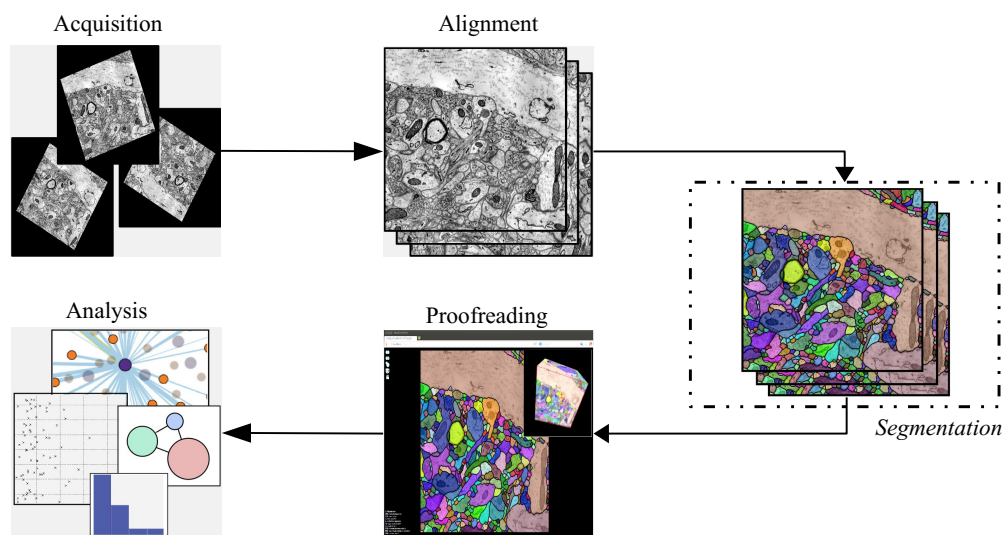


Figure 5: An illustration of the complete Connectomics pipeline from the Connectomics group at Harvard University, figure adapted from (Haehn et al., 2014)

In this pipeline, the brain images are acquired at nanometers-scale resolution using a Serial Electron Microscope (SEM). Then a series of image processing techniques are applied to these images through multiple stages of alignment, segmentation, and proofreading to prepare the data for analysis. Kaynig et al. (2015), published the complete reconstruction pipeline and demonstrated the first ever automatic reconstruction of individual spines in serial microscopy data prior to proofreading. Their seminal work performed segmentation by training a Random Forest (RF) classifier with sparse manual annotations of membranes collected interactively from users. The annotations enabled the classifier to correct the output in a feedback loop. They achieved significant improvement in automating the segmentation of membranes, and their results coupled with the proofreading stage have enhanced their overall Connectomics pipeline.

Since then, DNNs have been shown to outperform RFs for the task of segmentation of neuronal structures in EM images (Arganda-Carreras et al., 2015). In this thesis, we focus on the segmentation stage of the pipeline by combining the interactive approach of Kaynig et al. (2015) with a DNN model in a framework that trains a classifier using real-time distributed annotations of brain images.

Image Segmentation Techniques

In the field of computer vision, image segmentation is the process of partitioning an image into several regions with the goal of locating objects and boundaries in the image. In the past, the prevailing paradigm in computer vision is that of an object recognition relying on hand-designed feature descriptors that computer vision experts spend years crafting to extract useful information from pixels. This is usually followed by methods of pooling or histogram-

ming on features prior to classification. The only training would happen in the classifier itself such as in Support Vector Machine or a Bagging method like RF. Support Vector Machine is a machine learning classification method that performs decisions by constructing hyperplanes in a multidimensional space to separate objects of distinct classes (StatSoft, 2006). It supports both regression and classification tasks. Bagging is a technique used in ensemble learning algorithms to improve prediction accuracy. In bagging, the training data is distributed across multiple models and the results of prediction is computed by averaging the combined results of the models.

The question then becomes: how to get better features? The performance of these types of approaches plateaus after three or four years of people utilizing them with different variants of hand-designed features. Many of the traditional approaches to segmentation utilized simple local features for testing such as difference in pixels or regions (Schroff et al., 2008). More recent works, such as (Schmidhuber, 2012), utilized pixel-based techniques with deep neural networks to partition images.

In Connectomics, the task of segmentation involves partitioning neuronal structures in EM images of brain circuitry with the aim of reconstructing the structural and functional connections of the brain. Most approaches to segmenting neuronal structures distinguish between pixels that represent membranes which form the boundaries of cells, and pixels that form the interior of cells. The membrane pixels are typically darker than the pixels in the interior of the cells. However, depending on the imaging method, some intra-cellular structures are also as dark as membrane structures. Some studies corrected this problem by thresholding the gray-level EM image to separate light and darker pixels, and by applying filtering to enhance the membranes before or after thresholding (Jurrus et al., 2008) (Yang and Choe, 2009). However, this solution is not sufficient in most cases to produce high quality segmentations. As such, most research in Connectomics use machine learning techniques to classify pixels as

either membrane or non-membrane.

With the advent of deep neural network architectures, recent advances in computer vision have made dramatic improvements in image segmentation. The approach taken by this thesis leverages DNN hierarchical learning abilities. DNNs have seen wide application in computer vision in recent years with promising results in object recognition and classification tasks (Krizhevsky et al., 2012). The most notable DNN application is the recognition of handwritten digits in the MNIST data set (Ciresan et al., 2011) and the image segmentation work of (Schmidhuber, 2012). The MNIST data set consists of more than 60,000 examples of handwritten digits that the vision community has been using to benchmark performance of object recognition algorithms. In DNNs, the MNIST data set has been used to measure the performance of recognizing the digits of the decimal number system and it has achieved an error rate as low as 0.21 percent (Wan et al., 2013). The significance of the MNIST data set is that, when presented to a deep neural network, the network can learn to look for lines and loops when classifying the digits. The lines and loops are features that are learned through multiple-layers without human-crafted input representation. To understand how deep neural networks function, in the following section, we examine DNN techniques for image segmentation and the application of Random Forest(RF).

Deep Neural Networks

DNNs, also known as Artificial Neural Networks, are a class of machine learning algorithms that are loosely inspired by neuroscience. In neural anatomy, a biological neural network is composed of neurons which are connected via axon terminals to form pathways that allow signals of activated neurons to flow from one cell to another. This architecture is similar to that of DNNs in that, a DNN architecture is composed of highly-interconnected processing

elements organized into layers. The processing elements, referred to as units, perform computation in response to stimulation from external inputs. The computation generally involves passing a linear weighted sum of the input through an activation function that controls whether the neuron fires or not.

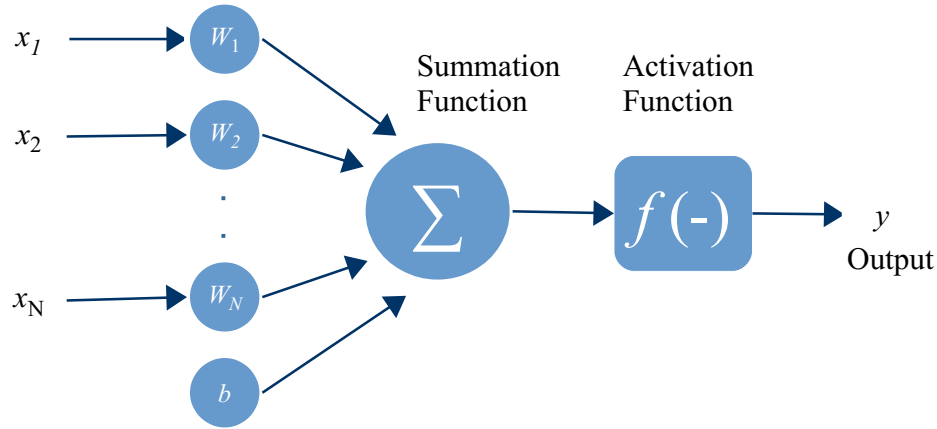


Figure 6: A single artificial neuron with N input vectors (x_1, x_2, \dots, x_N)

With respect to Figure 6, to illustrate how ANNs work, consider a supervised learning problem with training examples $S = \{ (x_i, y_i) \mid 0 \leq i < N \}$, where x_i is the i th observation (feature vector), y_i is the corresponding desired output vector for the i th observation, N is the number of training examples. In neural networks, we can define a complex, non-linear form of function f , with parameters W, b to fit the data; where W is a weight matrix corresponding to the input and b is a bias or threshold vector of the artificial neuron. Creating a network for this task may begin with a simple neural network consisting of a single unit as denoted in Figure 6. This single computational unit takes as input (x_1, x_2, \dots, x_N) , and produces the output

$$y = f\left(\sum_{i=1}^N W_i x_i + b\right)$$

where $f()$ refers to the activation function of the artificial neuron, which is described in the

next section.

This single-unit network operates by performing a linear transformation on the input by the weight matrix W . Each input x_i is multiplied with its corresponding weight W_i as it enters the unit. The inputs are weighted because some inputs are more important than others. Then a translation transform is applied by the bias vector b . The unit then sums the weighted input which produces an activation value that can be either negative or positive. The activation value is generated by a point-wise application of an activation function, described in the next section. If the activation value is larger than a certain threshold, the network outputs a signal, otherwise it outputs zero. This is the process by which a single unit makes decision based on input; and combining such decision units in a multilayer network enables the network to make complex decisions.

Activation Functions

Activation functions play a very important role in neural networks. Figure 7 shows the three most common activation functions used in neural networks.

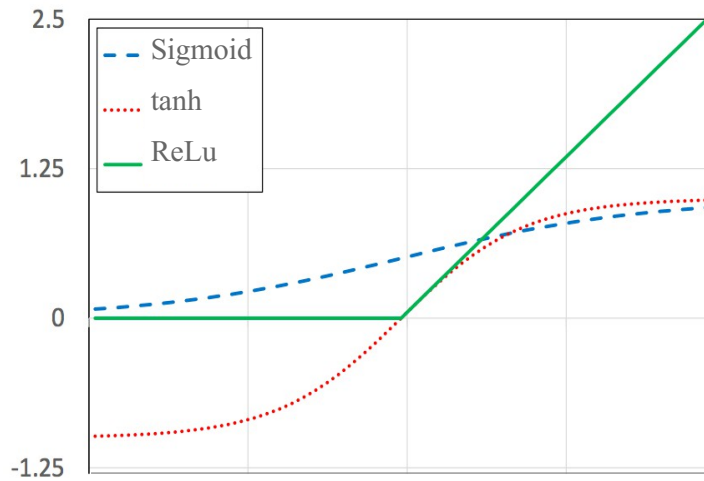


Figure 7: Common activation functions: sigmoid, tanh, and rectified linear.

The choice of these activation functions is dependent on their properties to enhance and simplify a neural network. For example, multilayer neural networks require non-linear activation functions in order to take advantage of the multiple layers of computation, otherwise, using a linear activation function without weight sharing would be equivalent to using a single-layer network.

In the literature, several activation functions exist that are commonly utilized in neural networks. These functions includes the logistic *sigmoid*, the hyperbolic tangent (*tanh*), and the rectified linear function (*ReLU*). Given z as the sum of the weighted inputs from Figure 6, the formulas for these functions can be defined as follows: $sigmoid(z) = \frac{1}{1+e^{-z}}$, $tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, and $ReLU(z) = max(0, z)$. Recent research have shown the rectified linear function to work better in deep neural networks (Glorot et al., 2011). Compared to *sigmoid* and *tanh*, the rectified linear function brings with it some key advantages. First, it eliminates the vanishing gradient problem of *tanh* and *sigmoid* functions by setting the gradient to 1 for every input that is greater than 0. The vanishing gradient is a problem encountered during DNN training with back-propagation algorithm. In back-propagation, the goal is to minimize the error of the objective function by finding its optimal local minimum. This process involves taking smaller steps in the negative direction of the objective function derivative with respect to the network gradients. In DNNs, the additional layers of computation causes the magnitude of the gradients to decay exponentially, thus resulting in slower learning of weights in the deeper layers of the network. The activation function is one of the contributing factors to the decay of the gradients. In both *sigmoid* and *tanh*, the magnitude of the derivative of the layers output with respect to input stays below 1.0 and makes the gradient vanish faster. Rectified linear function does not have this problem because the gradient is 0 for negative input and 1 for positive inputs. Secondly, the rectified linear function induces sparsity in the hidden units which plays an important role in the performance of the classi-

fier (Deng and Yu, 2014), and the computational costs are lower compared to the *tanh* and *sigmoid* due to the absence of exponential term.

Multilayer Perceptron (MLP)

Multiple units can be grouped together to form a multilayer neural network, as depicted in Figure 8, with one or more hidden layers between the input and output layers that are capable of learning abstract representations.

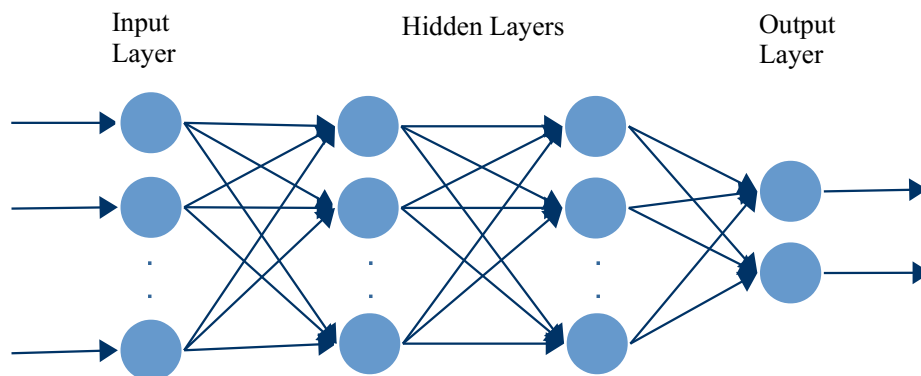


Figure 8: A deep neural network with an input layer, two hidden layers, and one output layer.

This type of network is known as a feedforward network because of the fact that output from previous layers is used as input to the next. The first layer, called the input layer, provides raw inputs to the the second layer. The middle layers, representing the hidden layers, are fully connected and are capable of extracting abstract representations (features). Typically an MLP model consists of one or more hidden layers with more layers making the network deep. A deep network is capable of extracting more features with higher abstractions from the input data. The final layer is the output layer which computes the prediction results for the

data. For classification tasks, the output layer is generally designed as a logistic regression layer that outputs class probabilities. The depth of a multilayer network has no restriction, as such, the addition of more layers can make the network a universal approximator; i.e., with the right parameters, a multilayer network can approximate any continuous function (Hornik et al., 1989).

For the sake of notation simplicity, we denote the input layer as layer 0 and the output layer as layer L for an $L + 1$ layer MLP. In the first L layers, we can compute the activation vector as follows:

$$v_l = f(z_l) = f(W_l v_{l-1} + b_l), \text{ for } 0 < l < L$$

Where $z_l = W_l v_{l-1} + b_l \in \mathbb{R}^{N_l \times 1}$, $v_l \in \mathbb{R}^{N_l \times 1}$, $W_l \in \mathbb{R}^{N_l \times N_{l-1}}$, $b_l \in \mathbb{R}^{N_l \times 1}$, $N_l \in \mathbb{R}$. z_l , v_l , W_l , b_l , and N_l are respectively the excitation vector, the activation vector, the weight matrix, the bias vector, and the number of units at layer l . $N_l \times 1$ denotes a one dimensional column matrix. Note that this formula works for $L - 1$ layers because the first layer, also known as the input layer, is just the observation vector, which is defined as $v_0 = x \in \mathbb{R}^{N_0 \times 1}$ and $N_0 = D$ is the dimension of the feature vector. $f(\cdot): \mathbb{R}^{N_l \times 1} \rightarrow \mathbb{R}^{N_l \times 1}$ represents the application of the activation function to the excitation vector element-wise. The output layer of the model needs to be chosen based on the task. For a regression task, the output layer is a linear logistic regression layer:

$$z_L = W_L v_{L-1} + b_L$$

that generates the output vector $v_L \in \mathbb{R}^{N_L}$, where N_L is the output dimension. Each output unit in the output layer represents a class $i \in \{1, \dots, C\}$, where $C = N_L$ is the number of

classes. The value of the i th output unit represents the probability $P_{mlp}(i|x)$ that the observation x belongs to the class i . The *softmax* function can be utilized to normalize the excitation to ensure a multinomial probability distribution.

$$v_i^L = P_{mlp}(i|x) = softmax(z^L) = \frac{e^{z_i^L}}{\sum_{j=1}^C e^{z_j^L}}$$

where z_i^L is the i th element of the excitation vector z^L . Therefore, given an observation vector x we can compute the output of the MLP model with parameters $\{W, b\} = \{W_k, b_k \mid 0 < k \leq L\}$ using the forward computation algorithm summarized in Algorithm 1.

Algorithm 1 MLP Forward Computation

```

1: procedure Forward-Computation
2:   for  $k \leftarrow 1; k < L; k \leftarrow k + 1$   $N$  do
3:      $Z_k \leftarrow W_k v_{k-1} + B_k$ 
4:      $V_k \leftarrow f(Z_k)$ 
5:   end for
6:    $Z_L \leftarrow W_L V_{L-1} + B_L$ 
7:    $V_L \leftarrow softmax(Z_L)$ 
8:   Return  $V_L$ 
9: end procedure

```

Training Process

Before the MLP model can interpret data accurately, it needs to be trained to adjust the weights and biases to fit the data. This means the parameters $\{W, b\}$ in the model need to be estimated from the training samples $S = (x_i, y_i) \mid 0 \leq i < N$, where N is the number of training samples, x_i is the i th observation vector and y_i is the corresponding desired output. Therefore, a training criterion is required that is highly correlated to the training task with the

goal of minimizing the training error as the parameters W and b are learned. The two popular training criteria in deep neural networks are the means square error (MSE) criterion for regression tasks, whose Jacobian matrix is defined by the equation:

$$J_{MSE}(W, b; S) = \frac{1}{N} \sum_{i=1}^N J_{MSE}(W, b; x_i, y_i), \text{ where}$$

$$J_{MSE}(W, b; o, y) = \frac{1}{2} \| v_L - y \|^2 = \frac{1}{2} (v_L - 1)^T (v_L - 1)$$

and the cross-entropy (CE) criterion for classification task, whose Jacobian matrix is defined by the equation:

$$J_{CE}(W, b; S) = \frac{1}{N} \sum_{i=1}^N J_{CE}(W, b; x_i, y_i), \text{ and}$$

$$J_{CE}(W, b; x, y) = - \sum_{i=1}^C y_i \log v_i^L,$$

y_i is the empirical probability observed in the training set that the observation x belongs to class i , and v_i^L is the same probability estimated from the model. In the case of a hard class label, where c is the class label for observation x , the CE reduces to a negative log-likelihood (NLL)

$$J_{NLL}(W, b; x, y) = - \log v_c^L,$$

Given the training criterion, we can utilize the famous back-propagation technique (Rumelhart et al., 1986) to learn the model parameters W, b and minimize the error on the objective function. With back-propagation, the differences in the output compared to the label is sent backwards through the network to compute the change in weights that corresponds to the gradient descend of the error function.

Learning is accomplished by iterating over the examples in the training set. One method of learning is called batch learning, whereby the gradient is estimated after a complete pass through the training set. Another method is Stochastic Gradient Descent (SGD) learning, whereby the gradient is estimated based on a random sample from the training set for each weight update. A method that we employ in this thesis is mini-batch learning with SGD. In the mini-batch approach, the gradient is estimated from a small number of examples from the training set. This method is very useful because it is faster than batch learning.

Because DNN models are highly non-linear and non-convex, the convergence of the gradient descent can be problematic as the gradient estimates may contain many local optima which are very poor. The SGD alleviates this problem by adding noise to the computation of the gradient estimates which helps it escape poor local optima. The concept of momentum can also be utilized to improve the speed of convergence of the learning algorithm(Bengio, 2012). Momentum temporarily smooths the gradient by applying a scaling factor to the learning rate and updates the parameters in the opposite direction of the gradient.

The training of DNN model can be performed until a stopping criteria is reached. The stopping criteria could either be when all training inputs produces the expected output or when a convergence criteria is met. A good heuristic for a convergence criteria is when the change in error over multiple weight updates is smaller than a set value.

Convolutional Neural Networks

Although MLPs are great at being universal approximators, their architecture suffer at processing visual information. One of the issues with MLPs is that the number of weights for the network connections grows rapidly as the dimension of input increases, thus slowing down training. Another problem is that MLPs do not take advantage of the spatial organization of

visual data. A variant of MLPs that is inspired by biological functions of the visual cortex, called Convolutional Neural Networks (CNN), was created to address these issues. The architecture of CNNs was first introduced by Fukushima (1980) and was later improved by LeCun et al. (1998). It addresses the shortcomings of fully-connected network by exploiting the properties of sparse connectivity and weight sharing.

Sparse Connectivity

CNN exploits the spatial correlation inherent in data such as 2-D images by enforcing local connectivity between units of adjacent layers. This means, the input of hidden units in layer l are connected to a subset of contiguous units in layer $l - 1$. These subsets are known as the kernels or filters whose weight are learned over time.

Shared Weights

Weight sharing is an important principle in CNN as it helps reduce the total number of trainable parameters during back-propagation training. It may also lead to a more efficient training and effective model when similar local structures repeat in the input space. In the CNN model, weight sharing is achieved by filters in the convolution layer. To understand this principle, we give a detailed description in the convolution operation next.

Convolution

The convolution operation is the most important operation of CNN models. It is applied by sliding a linear filter over the image in a small window, followed by an application of a non-linear function which produces a two dimensional matrix of activations, as depicted in Figure 9. Each filter consists of a weight matrix and a bias vector. Therefore, as the filters are slid through different positions of the input space to detect corresponding features, each

position shares the same weight matrix and bias vector. This process is illustrated in Figure 9, where a single filter of size $3 \times 3 \times 3$ is slid across an input image of size $5 \times 5 \times 3$ to produce an output of activations of size $3 \times 3 \times 1$.

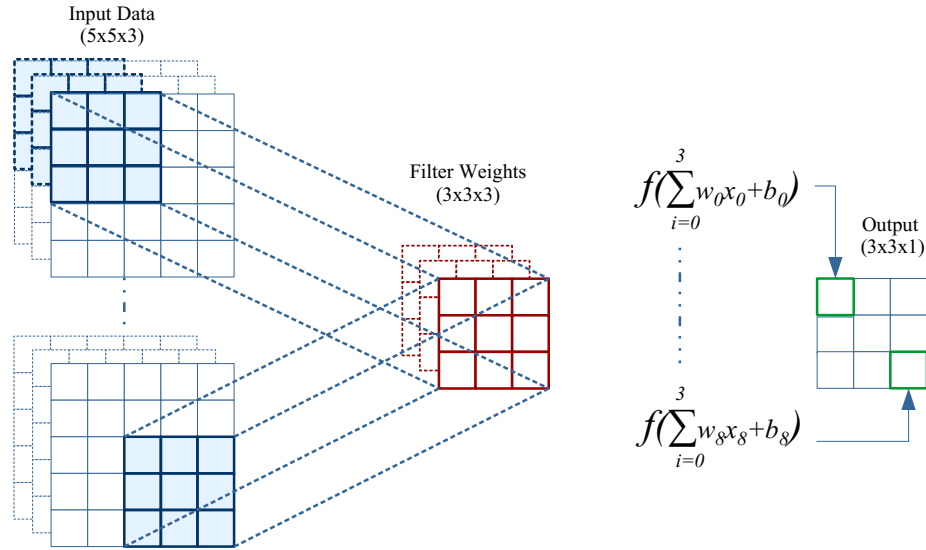


Figure 9: An illustration of weight sharing as a filter slide through the pixels of a $5 \times 5 \times 3$ input image.

In this example, the filter starts from the upper left portion of the image and slide left to right, top to bottom. Each slide operation, shown by the highlighted cells in the input data, computes the dot product between the input and the weight matrix, and the results is adjusted by the corresponding bias vector. The sliding operation is performed simultaneously on each depth slice of the image and the sum of all depths results into an activation output that is depicted as a single cell in the output matrix. To learn different features, multiple filters can be added and slid across the image in the same fashion as a single filter. Figure 10 illustrate another example of feature extraction based on a single filter.

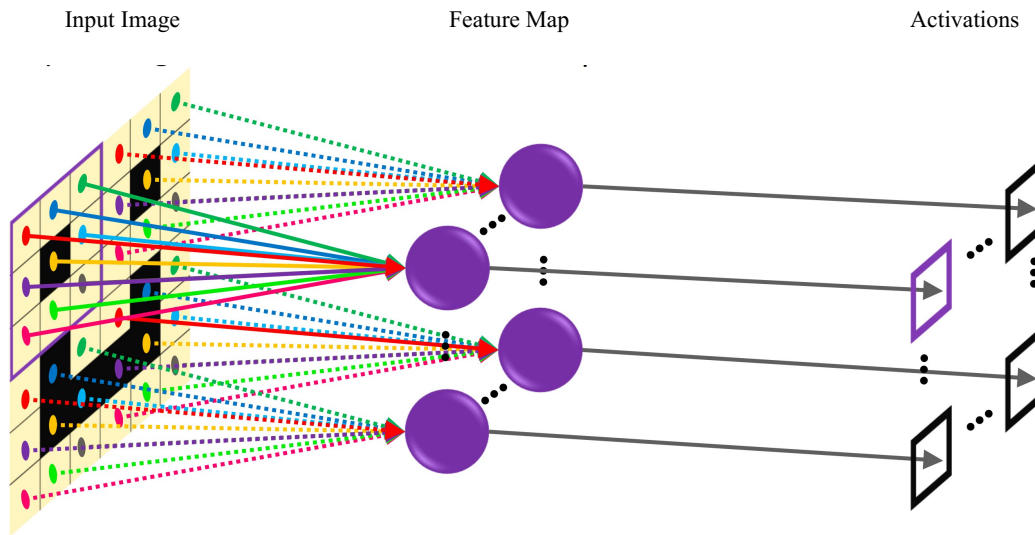


Figure 10: An illustration of a convolution layer creating one feature map in purple color, which performs a convolution operation on the image to produce an activation. Connections with the same color share weights. Figure adapted from Gudi (2015)

The convolution operation can be expressed mathematically as:

$$h_{ij}^k = f((W_k * x)_{ij} + b_k)$$

Where, h_{ij}^k is the $(i, j)^{th}$ value in k^{th} feature map. W_k and b_k are the weights and bias of the k^{th} filter, and f is an activation function. The net results of the convolution operation is that the network becomes aware of the spatial organization of the input data.

Max Pooling (Sub-sampling)

The other operation utilized by CNN to capture high-level features of input data is the max-pooling operation. The goal of the max-pooling operation is to reduce the overall size of

the signal by introducing robustness to variance in the input transformation. In max-pooling, the feature map is divided into a set of non-overlapping sub-regions and the maximum value in each sub-region is emitted. For a given 2-D patch, this operation evaluates the presence of a feature in the area of the input space.

The architecture of a CNN, depicted in Figure 11, is comprised of three types of layers: convolution, max pooling, and fully connected.

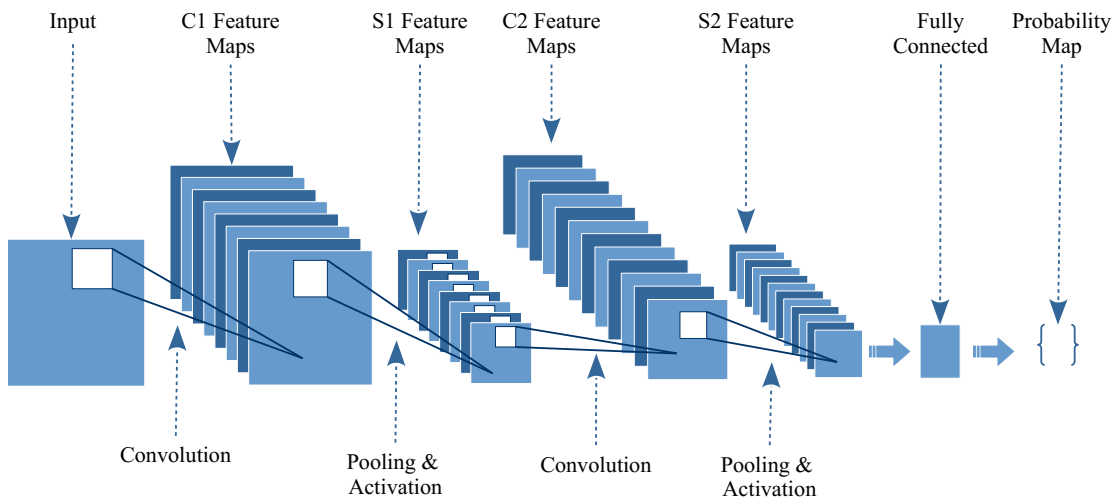


Figure 11: An illustration of a CNN model with 2 layers of convolution, 2 layers of max pooling, and 1 fully connected layer.

The convolution and max pooling layers are applied one after another followed by one or more fully connected layers (MLP). The fully connected layer is optional and cannot be followed by a convolution or max pooling layer. The input layer expects data in 3 dimensions of $w \times h \times d$, where w is the width, h is the height, and d is the depth. For an image, d is the number of color channels.

Application

DNN models have become increasingly popular in recent years for solving classification problems as a result of their high-accuracy rate. Their applications extends beyond image classification into financial analysis, speech recognition, Natural Language Processing, and many more fields. In pattern recognition, deep neural network methods are now the state of the art. In ImageNet classification competitions, deep neural network models now dominates with a classification error as low as 5% compared to 25+% error rate in traditional methods (Russakovsky et al., 2015). In comparison to traditional statistical machine learning methods which require a human domain expert to construct a good set of features as input, deep neural network models do not require hand-designed features to begin learning. Therefore, they're well suited for Artificial Intelligence (AI) tasks such as visual object classification and speech recognition.

Of the many approaches of deep neural networks, the most significant and similar approach to this thesis is the work of Ciresan et al. (2012) in segmenting EM images. In Cire-san et al. (2012), a DNN classifier is used to perform segmentation of EM images based on manually annotated membranes in a training set. The classifier is build as a CNN model that computes the probability of whether a pixel is a membrane or not based on the intensities of training sample images. The input layer of the model maps each raw input pixels to a unit, followed by a series of convolution and max-pooling layers which preserve 2-D information and extract features with higher level of abstraction. The output layer of the model produces a probability for each class. The resulting probability map is used to segment the entire image by classifying all of its pixels. The training of the model utilized a set of manually-annotated images with ground truth and the prediction was performed on a similar-sized set of unlabeled images without ground truth. The final output of the model is passed through a post-

processing step to finalize the resulting segmentation.

In spite of the major advantages of deep neural networks elaborated above, there are some weaknesses. One of the key weakness of deep neural networks is their tendency to overfit due to the additional layers of abstraction. A mitigation to this problem is to apply regularization techniques to ensure they generalize to new cases. Another weakness is that the training of deep neural networks requires a large amount of data compared to traditional machine learning methods. The large data-set requirement and the scale (number of layers) of these models directly impacts their performance and accuracy.

Random Forests (RFs)

Many random forests approaches have been applied to the task of segmenting images, most notably the above-mentioned work of (Kaynig et al., 2015), and the work of (Sommer et al., 2011) on ilastik. Ilastik is a standalone software tool that combines machine learning tools with a graphical user interface to train a classifier for image segmentation. In ilastik, a random forest classifier is trained and fine-tuned based on interactive input from users (Sommer et al., 2011). A user enters labels into the tool and paint the surface of the input image to mark corresponding pixels to each label. The random forest classifier then trains itself with the data and provides feedback to the user for the next set of labels. The fully-trained model is then used to automatically process a very large set of images in a batch processing mode. The training of the decision trees in ilastik utilized a bootstrap sampling methodology, whereby the training data is sampled into equal-sized sets with replacement. During prediction, each pixel is classified by aggregating the votes of each individual decision tree. The ratio of the votes is interpreted as a probability that forms the basis of the segmentation task.

Outside of image segmentation, Random Forest has seen a lot of success. In Kag-

gle machine learning competitions, a number of the winning projects utilized Random Forest techniques. For example, the winning entries for the classification of malicious URL utilized several variants of Random Forests, which were trained with labeled data and features selected from supervised and unsupervised methods. In finance, Theofilatos et al. (2012) used Random Forest technique to model trading using the EUR (Euro) and USD (dollar) exchange rates. Their project focused on predicting the one-day-ahead movement direction of the EUR-USD exchange rates. They achieved satisfactory results which outperformed other methods such as K-Nearest Neighbors, Naive Bayesian classifiers and Support Vector Machines.

Random Forest classifiers excel in computation speed and their ability to deal with unbalanced and missing data. Their main weakness is the inability to predict beyond the range of the training data, and over-fitting on noisy data-sets. In terms of application, Random Forest methods are well-suited for analysis of complex data structures embedded in small to moderate size data sets.

Example of other machine learning techniques that have been used in identifying neurons in EM images includes: k-means clustering (Lucchi et al., 2010), support vector machines (Glasner et al., 2011) (Lucchi et al., 2010)), and boosted decision stumps (Venkataraju et al., 2009).

Data Collection and Curation

The data used in this thesis was acquired from Dr. Lichtman's Connectomics group at Harvard University. The data comes from sections of a tissue of a dense mammalian neuropil from layers 4 and 5 of the S1 primary somatosensory cortex of a 5 month old healthy C45BL/6J mouse. The original data set captures a $30 \times 30 \times 30 \text{ } \mu\text{m}^3$ volume of tissue which

was taken by Electron Microscopy at a resolution of 3 nm per pixel and down-sampled by a factor of two to a resulting image plane of 6 nm per pixel (Kaynig et al., 2015).

We use a small subset, consisting of 240 gray value section images of brain circuitry, each of size 1024×1024 pixels. Each of the gray value images has a corresponding image for membrane labels, background labels, and ground truth labels. 210 of the gray value images and corresponding labels are used for training, twenty are used for validation, and ten for test. The membrane and background labels were created by biologists and are used for evaluation purposes only. The membrane labels are used to extract membrane examples from the gray value images. The background labels are used to extract background examples from the gray value images. The ground truth labels are used to generate variation of information.

In interactive mode, we only use the gray value images to capture annotations from users, which we use to train the model and visualize segmentation results. In offline mode, during evaluation, we use all 240 gray value images with corresponding membranes and background labels. We first train the model with the gray value images and corresponding membrane and background labels in the training set, then we use the test set to generate segmentation results. From the segmentation results, we generate variation of information to measure the performance of the trained model.

Summary

In this chapter, we gave an overview of neural anatomy and its relevant components to Connectomics. We described how the synaptic connectivity between neurons of an organism forms the connectome of the nervous system of that organism, and how neuroscientists believe the connectome could enable them to better understand the functions of the brain.

We then gave a detailed background of the state of the art methods of image segmentation currently used in Connectomics, paying special attention to deep neural networks methods, which are employed in this thesis. Then, we concluded the chapter with a description of the data used in this thesis and how it is collected and used. In the next chapter, we present our approach to image segmentation based on the deep neural networks methods described in this chapter.

Chapter 3 Interactive Segmentation

In this chapter we present our approach to image segmentation by formulating an interactive framework based on deep neural networks techniques. We present a detailed architecture of our framework, outlining the process of annotating pixels, the process of training a classifier based on deep neural networks techniques, and the real-time feedback loop by which annotations flows from a user to a running model and segmentation results flows back to the user.

Approach

Our approach to segmentation exploits the power of DNNs by creating an object detector which we utilize to detect membranes and other organelles in a cell. We create the detection module in a framework that provides real-time interactivity and flexible interface for configuring learning algorithms, capturing training input, and visualizing segmentation results. We employ a project-based system that defines one model per project. A single project encapsulates the hyper-parameters of a model, the class labels of objects to be recognized, and the set of training images that can be annotated interactively. This segregation enables advanced users to quickly configure, test, and compare network architectures with a small but representative sample of the training data and obtain a reference configuration that can be use for a task. Central to the system is the interactive training of the model. We capture annotations from users to train the model, then use segmentation results to visually guide the users where

to provide new annotations. This feedback loop occurs in real-time and enables us to reduce the manual effort required to fully annotate an image. Once a model is fully trained, it can be disconnected from the framework and use to segment images offline.

System Architecture

The overall architecture of our system is depicted in Figure 12, consisting of a user interface , a web service, a data store, and a deep neural network module.

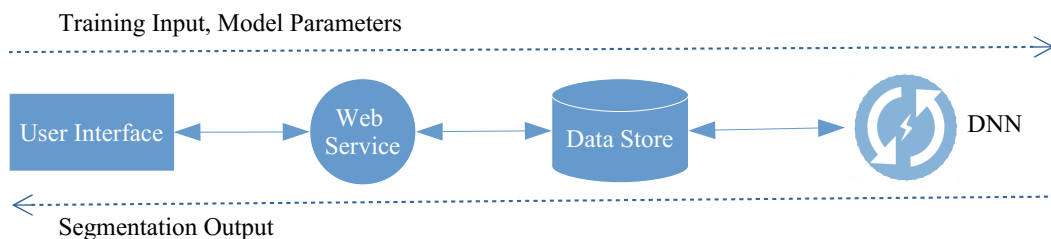


Figure 12: The architecture of the system consisting of a user interface, a web service, a data store, and a DNN framework.

Training input and model configuration flows from the user interface to the DNN module and segmentation output flows from the DNN module to the user interface. The user interface runs on a web browser and connects to a web service to access the system. The web service and DNN module runs on a Unix system as separate processes. The data store links the web service and DNN modules by providing a data exchange hub where both modules can read and write information. All the components of the system are highly-decoupled and are designed to target level 5 of the Capability Maturity Model (CMM) of service oriented architecture to ensure a failure on one component does not impact the entire system. As such,

any of the system's components can be disconnected without affecting the usability of the system.

Project

The basic operating unit of the system is a project, shown in Figure 13. A project encapsulates the configuration settings necessary to train a DNN classifier for image segmentation. It consists of the model parameters, the class labels of objects to be recognized, and the set of training images.

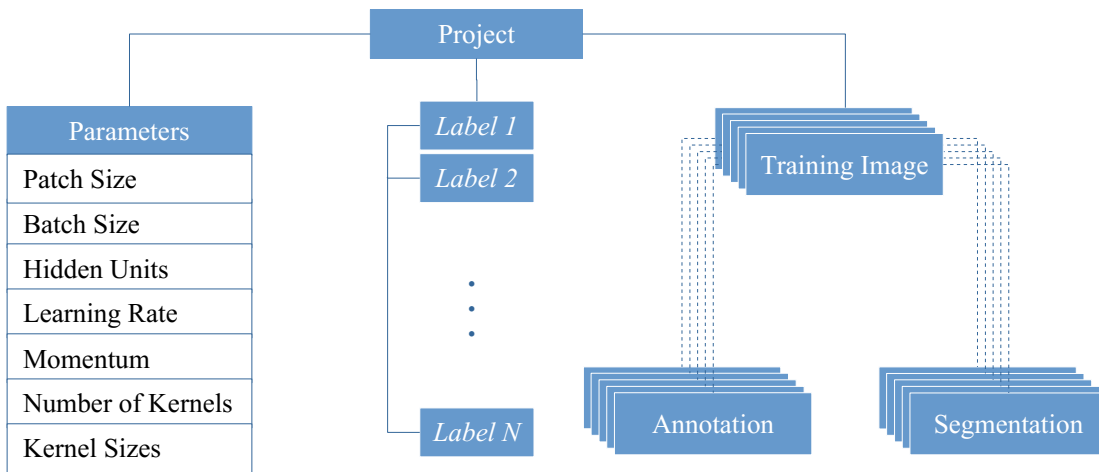


Figure 13: The anatomy of a project consisting of a model's parameters, class labels, training images, annotations, and segmentations.

In this revision of the software, we maintain a one-to-one relationship between a project and a DNN model. The implication of this design decision is that annotations cannot be shared across DNN models; however, we do provide the ability to copy data and configuration from an existing project to a new project. We may provide the annotations sharing

capability in a future revision of the software.

For persistence, the project information is written to a SQL Lite database by the web service module and is read by the DNN module. We designed the project to be configured by advanced users who are knowledgeable in creating and tuning DNN algorithms. Arbitrary number of projects can be defined but only one project can be active in interactive mode. By grouping data by project, we are able to address data integrity issues and implement multi-class segmentation.

Data Integrity

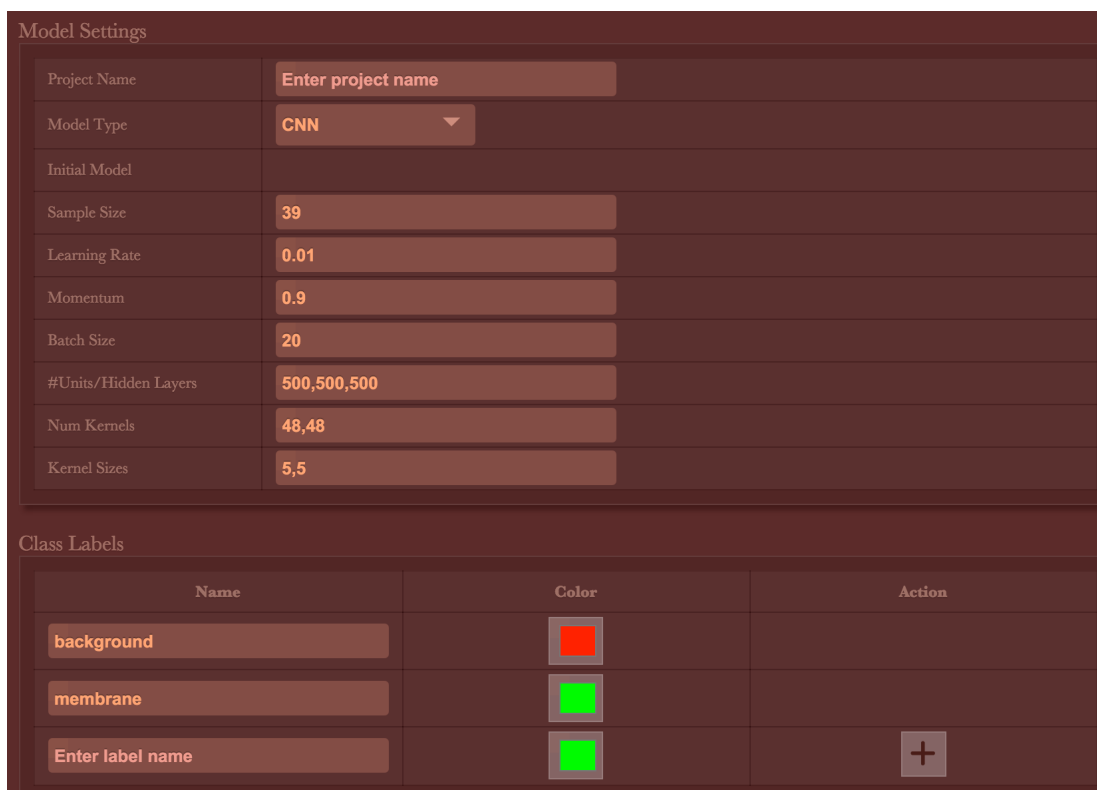
Because of the distributed nature of the system, we want to ensure that data provided by one user is not trampled by another. To this end, we provide the ability to create projects with unique names, and the ability for users to select a project before editing annotation data. All data associated with a project is stored with the unique identifier of the project. For annotation editing, we impose a limit of one user per image by instituting a locking mechanism which is described in the "Distributed Annotations" section.

Multi-Class Segmentation

Within a project, we allow definition of arbitrary number of object class labels. Each label is associated with a tag name, a numeric identifier used for training categorization, and a color used for visualization. During the annotation process, the labels are used to mark pixels on the surface of an image forming the feature set for training the model.

Project Editor

The project information is captured through a single user interface with two parts. The first part, depicted in Figure 14, captures the type of DNN model, its hyper parameters and the set of object classes to be detected. We provide a choice of two models in this version of software: MLP and CNN. Object classes are defined by entering a name in the text box, choosing a color by clicking on the color button, and adding the class to the project by clicking the add button. We allow arbitrary number of object classes to be defined. We recommend that the user choose complementary colors for the class labels so that visualization of annotations and segmentation is distinguishable.



The screenshot displays the Project Editor interface, divided into two main sections: Model Settings and Class Labels.

Model Settings: This section contains a table of configuration parameters for the model. The parameters and their values are as follows:

Parameter	Value
Project Name	Enter project name
Model Type	CNN
Initial Model	
Sample Size	39
Learning Rate	0.01
Momentum	0.9
Batch Size	20
#Units/Hidden Layers	500,500,500
Num Kernels	48,48
Kernel Sizes	5,5

Class Labels: This section is a table with three columns: Name, Color, and Action. It lists the classes to be recognized and provides a way to add new ones.

Name	Color	Action
background	[Red color swatch]	
membrane	[Green color swatch]	
Enter label name	[Green color swatch]	[+]

Figure 14: A project editing user interface that captures configuration parameters of a model and the class labels of objects to be recognized.

The second part of the project editor, depicted in Figure 15, captures the set of images for training and validating the model. We provide two multi-select lists for each set, one consists of the set of available images, and the other consists of the user-selected images. The arrow-buttons are used to move images between the lists.

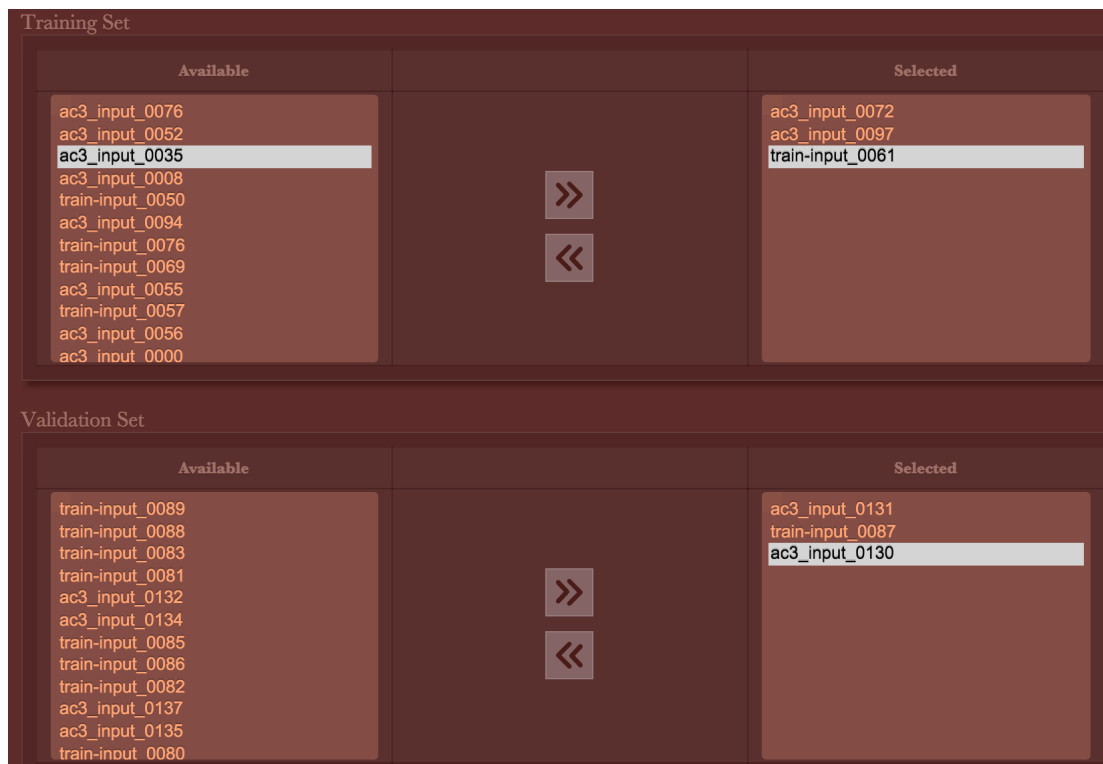


Figure 15: User interface screen for capturing training and validation sets.

We utilize the same interface for editing an existing project with non-tunable values locked. We perform client-side validation of input entries before transmitting to the server. Entries containing errors will prompt user to correct mistakes. We store the project information in a SQL Lite database that the DNN framework reads to perform its training and segmentation tasks. Running a project is as simple as activating it from the Image Browser screen which is described in the Image Browser section.

DNN Framework

The architecture of our DNN framework is illustrated in Figure 16, consisting of training and segmentation modules running in parallel on separate processes. We employed this architecture in order to fast-track the interactive feedback loop so that the training and segmentation tasks can be performed on separate GPUs and possibly separate computing nodes.

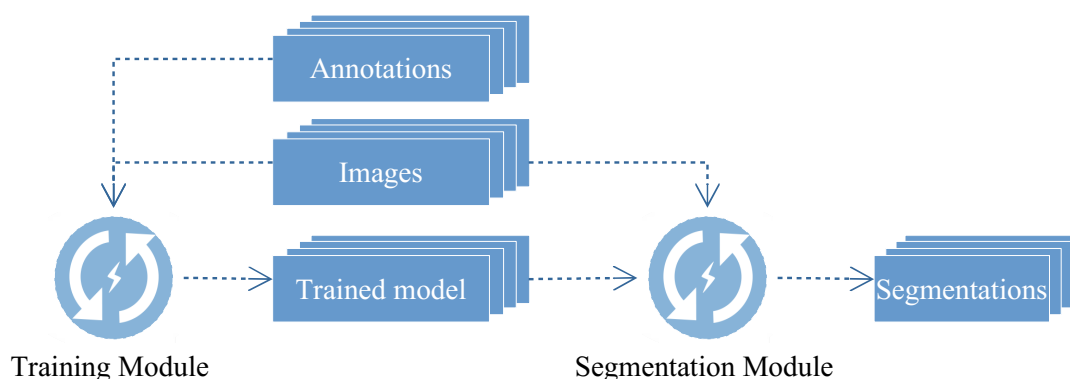


Figure 16: The execution model of the DNN framework consisting of training and segmentation modules running in parallel.

We create our DNN framework to support MLP and CNN models described in chapter 2. The architecture of the MLP model consists of one input layer, one output layer, and an arbitrary number of hidden layers. We expose the number and size of the hidden units as configuration parameters. For CNN, the architecture is comprised of an input layer, one or more combination of convolution and sub-sampling layers, and a fully connected layer that contains a logistic regression output layer. The number of kernels and kernel sizes of the CNN model are exposed as configuration parameters to be set by user on creation.

We implemented the DNN framework in Python programming language and utilized the Theano library (Bergstra et al., 2010). The Theano library offloads most of the work of

computing gradients and Hessian matrices, and the compilation of code into optimized C or CUDA for fast computation. It makes the development process easier and enable us to focus on symbolic expressions as opposed to low-level code. To support parallel execution of the modules, we used Python's multithreading to spawn a thread for each module when a project is activated.

Execution

The framework operates on one active project at a time in interactive mode. In offline mode, the framework relies on a user-specified configuration for model initialization and input and output data paths. This mode is useful when evaluating performance of a classifier or when segmenting a set of images based on a fully trained classifier. In interactive mode, the framework periodically checks the database for an active project. An activated project is launched by reading its configuration from the database and instantiating its training and segmentation modules.

The training module runs in a thread loop that executes one epoch of training followed by a check for new training data on each iteration. During the training epoch, we use sample randomization to perform mini-batch training based on the set of user annotations. We periodically compute and track a validation error, which we use to measure the learning ability of our model. When the validation error improves, we store a copy of the model to disk and notify the segmentation module by writing a status to the database.

The segmentation module, similarly, runs in a thread loop that checks for new serialized model and images to classify. When a new model is available, it deserializes it, retrieves the list of images to classify, then performs segmentation on each image. The results of segmentation is written to disk and a segmentation status is written to the database. The web ser-

vice checks the segmentation status to retrieve the segmentation data and transmit it to the UI for visualization. The list of images to be segmented are prioritized so that those being annotated or viewed have a higher segmentation priority.

Model Initialization

At initialization, we check for a serialized version of the model. If one exists, we initialize the model's weights and parameters based on the deserialized version. Otherwise, we initialize the weights by randomly sampling from a uniform distribution. This is important because of the symmetric and interchangeable nature of neurons in DNNs. If the weights were initialized to the same value, all the neurons of the hidden layer would output the same value and detect the same feature patterns in the lower layers. The random initialization breaks the symmetry (Bengio, 2012).

Sample Randomization

One of the challenges of interactive segmentation is achieving an optimal speed that provides users with useful feedback while training is performed in the background. This challenge applies across all the components of the system but is particularly important for the DNN model because useful feedback to user can only be provided if the model is able to learn quickly and produce segmentation in a reasonable amount of time.

One of the strategies we employed is to develop a good sampling criteria that allows the DNN model to quickly learn and provide useful feedback to users. Our sample selection criteria involves drawing a maximum of one million examples from the list of images in the training set. The samples are drawn evenly across images and labels. Because our training data is grouped by class, we draw the examples randomly and also shuffle the drawn exam-

ples in order to avoid biasing the gradient during training. We also perform rotation on the samples to ensure the network can recognize objects in different orientation. During training, we draw a large batch from the examples which we submit to GPU; then we perform mini-batch training over the large batch. After each epoch, we re-sample the data by focusing on the badly-performing examples. Our heuristic for judging the performance of the examples is by examining the error between the network output and the target value. A large error indicates the input has not been learned by the network, therefore we need to present this input more frequently. Prior to each training iteration, we split the training batch based on the results of the previous iteration. Then we retain a maximum of 50% of the bad performing examples and draw the rest from the randomized pool. As the network trains, the relative errors will change as well the frequency of presenting a particular input. This technique is particularly useful in interactive training as it allows the model to quickly learn and provide useful segmentation results to guide annotation. Additionally, we prioritize new training over old training data to ensure that the learning algorithm is able to see interactive input and quickly provide feedback.

Reproducibility and Restartability

Because of the random-nature of parameter initialization and training, we institute a checkpoint system where periodically we serialize the best model to disk so re-initialization can take advantage of the learned parameters. The serialization stores all the necessary information including model parameters, current random number generator, parameter gradient, momentum, and so on. We use the serialized data for initialization as well as segmentation purposes.

Hyper Parameters

Our DNN models are controlled by a set of configurable parameters that determine their architecture, performance, and how they're trained. Following is a description of these parameters and the common techniques for setting their values.

Patch Size

This parameter controls the dimension of the feature space, i.e., the number of input units in the DNN model. For CNN, it determines the size of the feature maps. We use the patch size as the dimension of the training patches and it must be an odd number to enforce symmetry. For neuronal structures, we used a value between 29 and 65. This number is set once on model creation and is not tunable because it requires recreating the layout of the model; as such we provide the ability to copy an existing project to a new project whose parameters can be modified. In a future revision, we may expose tuning when we have a good mechanism for restructuring the model when patch size changes.

Batch Size

This parameter defines the size of the mini-batch we use for training the DNN model. It is the number of examples that are processed in parallel on the GPU. The choice of the batch size directly affects the convergence speed and the nature of the resulting model. We use mini-batches, randomly drawn from training examples, in combination with the SGD algorithm to estimate the gradient of the parameters of the learning algorithm. Our mini-batch size is the same throughout training and is set at model creation. Because the mini-batches training approach permits parallelization within the mini-batches, it greatly improves the convergence speed of the SGD algorithm.

Learning Rate

The learning rate is a factor of the SGD algorithm that we employ for training our deep neural networks models. It determines the rate at which the model learns and how quickly it converges. In gradient descent, given the parameters of the system represented by θ , our goal is to minimize an objective function $J(\theta)$ parameterized by the model's parameters by updating the parameters in the opposite direction of the gradient of the objective function $\Delta J(\theta)$ with respect to the parameters. This is accomplished in steps by following the direction of the slope of the objective function's surface until a valley is reached. The size of each step taken is determined by the learning rate. In our implementation, we use mini-batch SGD, which performs an update for every mini-batch n of training examples as follows:

$$\theta = \theta - \gamma \cdot \Delta_{\theta} J(\theta; x_{i:i+n}; y_{i:i+n})$$

where γ is the learning rate, x is the training examples, y is the labels, i is the index of the training examples, and n is the size of the mini-batches.

We expose the learning rate parameter to be determined by an expert user when creating a model. A good way to determine the correct learning rate is to perform experiments in conjunction with other parameters using a small but representative sample of the training data. This approach makes it possible to obtain a reference configuration from which a model can be based. Another approach is to determine the learning rate empirically based on the training batches. In this approach, we start with a set batch size and a learning rate, then after training a few hundred batches, we reduce the batch size, learning rate, or both until the training criteria is improved. The latter option is not available in this version of the software. A UI field is provided in the project configuration screen to capture the learning rate.

Momentum

One of the problems of SGD algorithm, which occurs around local optima points, is that it has difficulty navigating ravines, i.e, areas where the gradient surface curves more steeply in one dimension than in another (Sutton, 1986). This causes the SGD estimate to oscillate across the slope of the ravines and get stuck in local minimum or saddle point. One of the methods for improving this problem is the momentum method. The goal of momentum is to help accelerate the SGD by dampening the oscillations along the ravines slope. Therefore, given the update vector v_t at time t , the momentum method adds a fraction α of the previous update vector to the current update vector:

$$v_t = \alpha v_{t-1} + \gamma \cdot \Delta_{\theta} J(\theta)$$

$$\theta = \theta - v_t$$

However, the momentum method follows the gradient blindly without a notion of where its going or when to slow down. As such, we use Nesterov's accelerated gradient algorithm (Rossum, 1983), which updates the model parameters by calculating the gradient with respect to a future position of the parameters instead of the current parameters. The momentum equation in Nesterov's update is as follows:

$$v_t = \alpha v_{t-1} + \gamma \cdot \Delta_{\theta} J(\theta - \alpha v_{t-1})$$

$$\theta = \theta - v_t$$

We expose the momentum parameter as a configuration option in the UI for advanced users to update, and we set its default value to 0.9 based on our experiments. We recommend this parameter to be updated in conjunction with the learning rate. A high momentum parameter can help increase the speed of convergence of the system. However, setting the momentum parameter too high can create a risk of overshooting the minimum, which can cause the system to become unstable. A momentum coefficient that is too low cannot reliably avoid local

minima, and can also slow down the training of the system.

Network Architecture

We provide the network architecture, the number of layers and units per layer, the number of kernels, the sizes of kernels, as hyper parameters for constructing the learning model. These parameters are configured and tuned in the project configuration user interface described in Figure 14. Because each layer acts as a feature extractor of the previous layer, the number of units at each layer is recommended to be large enough in order to capture the essential patterns, but small enough to prevent overfitting. Overfitting is a very common problem in machine learning. It occurs when a complex model is said to not generalize well, i.e., it performs well on training data and poorly on new data. Machine learning experts deal with this problem frequently by making trade-offs between overfitting and model complexity. If a model is not complex enough, it may not be able to capture essential features to perform a task; on the other hand, overfitting can occur if the model is too complex. Some techniques in the machine learning community for fixing this problem include regularization, also known as weight decay, which augments the objective function by introducing terms that penalizes large weights so that the parameters do not fit the training data too well. Another technique is dropout, which performs random omission of units in each hidden layer for each sample during training. This means the remaining units need to perform well in the absences of the dropped-out units; and causes each unit to depend less on other units to detect patterns. For our MLP model, we used three hidden layers of 500 units. For our CNN model, we used two layers, each consisting of 48 filters of size (5×5) , followed by a fully connected layer with one hidden layer of 200 units.

Distributed Annotations

Distributed annotations is the process by which multiple users annotate distinct images in parallel while a running DNN model aggregates the annotations to train a classifier in real-time. This process is facilitated by a rich graphical user interface that runs on a web browser and a web service process that handles concurrent connections and mediates the exchange of data between the UI and the DNN model. We chose web technologies for implementing the user interface because of the cross-platform nature of web browsers. The entire process and data flow of annotating an image is depicted in Figure 17 for a single user. For distributed annotations, this workflow is replicated.

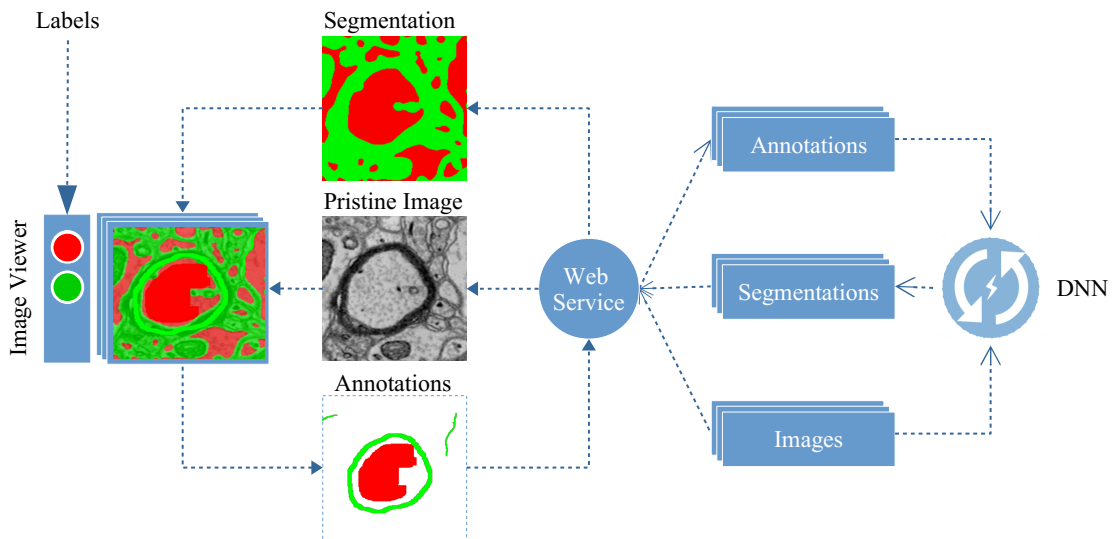


Figure 17: An illustration of the real-time interactive feedback loop of annotating an image while a DNN model train in the background.

Figure 17 shows the real-time feedback loop of annotating an image while a DNN classifier trains and guides the user. The image viewer consists of three visualization layers. The bottom layer renders the gray value image. The middle layer renders segmentation results

in the corresponding colors of the object classes. The top layer renders the annotations for each object class. For implementation, we maintain each layer as an off-screen render target that is not visible to the user. All three layers are flattened into a single image when rendering to the screen. This mechanism improves the performance and responsiveness of the UI.

Data Synchronization

For real-time updates, we maintain two synchronization timers: one for transmitting annotations from the UI to the server, and the other for transmitting segmentation results from the server to the UI. The timers are accompanied by a progress bar indicator which is rendered in the status area of the image viewer. The status bar is described in the Annotation Editing section. The annotation synchronization time is set by the application at installation time and is not tunable. The segmentation synchronization time is determined by the segmentation module as the amount of time required to fully segment a single image, and it can vary depending on the performance of the segmentation module. By calculating the time in the segmentation module we're able to present an accurate status to the user.

Annotation Lock

To prevent multiple users from concurrently modifying an image's annotations, we institute an exclusive lock that associates a unique identifier with the browser session of the first user to annotate the image. The lock is guaranteed as long as the user is modifying annotations and the exclusive timer hasn't expired. The expiration time of the lock is thirty minutes of idle time - a period in which no changes to annotations is detected. We manifest the lock by excluding editing tools from the image viewer screen of all users except the user with the exclusive lock.

Image Browser

To facilitate distributed annotations, we provide the image browser, depicted in Figure 18, which allows users to select an image and launch it to an annotation editor.



Figure 18: The image browser or home screen of the application showing the performance metrics, training set, and validation set of a selected project.

The image browser consists of a drop down list of available projects. For each project, the image browser provides a set of tools for editing, a display of its performance metrics, and a list of its training and validation images. The performance metrics shows the training and validation errors of the learning model in real-time. The images are shown as a grid of thumbnails with a color showing the status of the image. A gray color shows an image that has neither been annotated nor segmented. A blue color shows an image that has been segmented but

not annotated. A green or orange color is used on training images that has been annotated and scored. The score is computed as the error between the network output and the target value, with low error signified by green color and high error signified by orange color. We also render a locked key icon on the thumbnail of an image if it is being annotated by another user. When a thumbnail is clicked, we launch the annotation editor which is described in the next section. We require a minimum of one image from training set and one from validation set to be sparsely annotated before training can be performed.

Annotations Editing

The annotation editor, depicted in Figure 19, is the core interface for capturing annotations and visualizing segmentation results. It is comprised of a tool bar, an image view, and a status bar.

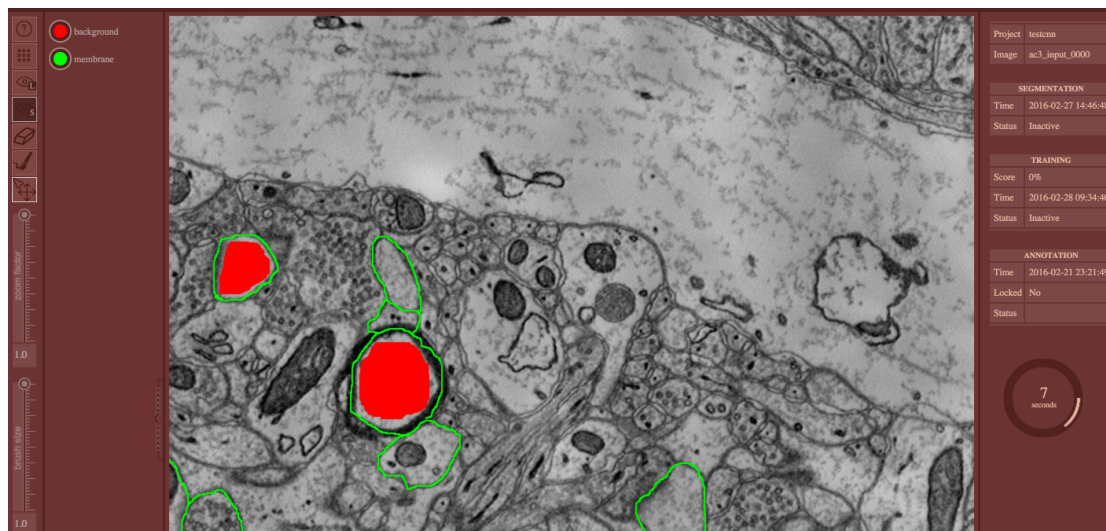


Figure 19: The annotation editing UI showing annotations for two labels: a background label in red and a membrane label in green.

The tool bar contains the set of tools for editing annotations and manipulating the

image view. These tools include: a brush for adding annotations, an eraser for removing annotations, a scale for controlling the size of brush or eraser, a scale for zooming in and out of the image view, a button for hiding or showing annotations, a button for hiding or showing segmentation results, a button for returning back to the image browser, and a button to launch the associated help screen.

Next to the toolbar is a panel that displays the list of object class labels rendered in their corresponding colors and names. The labels are presented as radio buttons so that only one label can be selected at a time. The selected label is drawn with a white border outline on its radio button. The panel hosting the labels is dynamic in that it can be expanded to reveal the names of the labels or collapsed to only show the radio buttons.

The image view portion of the user interface is rendered in three layers: the background layer is the gray value image, the middle layer is the segmentation results, and the foreground layer is the annotations. For validation images, the background layer is rendered with an overlay of the word "Validation Image". The top two layers are rendered with transparency so as to correctly guide the user during the annotation process. Annotations are added by selecting the brush tool and corresponding label, then painting on the surface of the layers to mark the pixels of the corresponding object. Removal of annotations follows the same process but with the erase tool selected. Changes to annotations are transmitted to the server via the web service and stored in a repository for the DNN model to read.

The final piece of the image viewer user interface is a status panel. This panel shows information about the current image and the status of the training and segmentation modules. It also contains a progress bar showing the remaining time until the next segmentation pull request.

Segmentation Visualization

Figure 20 shows the visualization of segmentation results rendered as an overlay on the gray scale input image, and as a background to the annotations layer. To retrieve segmentation results, we send a pull request to the server periodically based on the segmentation synchronization timer discussed in the Data Synchronization section. The results of segmentation is transmitted to the Image Viewer by the web service and is rendered in the segmentation layer of the image view.

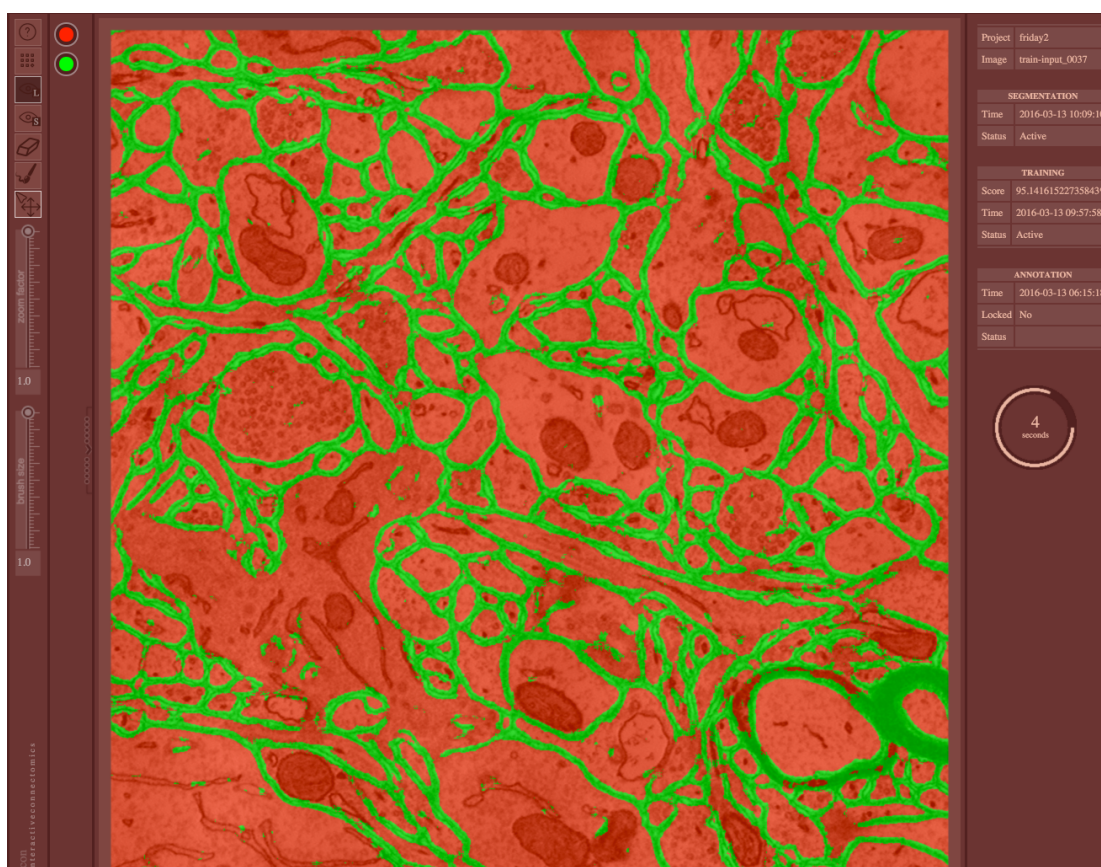


Figure 20: Results of segmentation rendered as overlay on gray value image.

Web Service

To facilitate the interactivity between users and the DNN model, we provide a web service, depicted in Figure 21, that accepts training input from the UI and writes data to a central repository for the DNN model to read. Conversely, the web service reads segmentation results from the central repository upon synchronization requests from the UI and writes back to the UI. We achieve this with a web server technology called Tornado (Dory et al., 2012). Tornado is a web server that provides asynchronous networking capabilities and non-blocking input/output that enables dynamic communication between users and the web service over Hyper Text Transfer Protocol (HTTP).

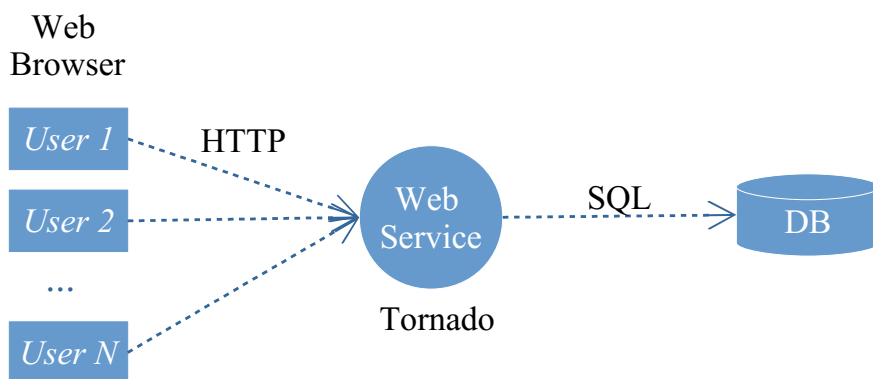


Figure 21: An illustration of the web service architecture showing multiple users interacting with the system over HTTP and the web service interacting with a database over SQL.

Because of the distributed nature of the web service, we're able to implement the distributed annotation of images where multiple users can annotate distinct images in parallel with the same DNN model running in the server.

Central Repository

We store all training and segmentation data in a central repository composed of a file system and an SQL Lite database as shown in Figure 22.

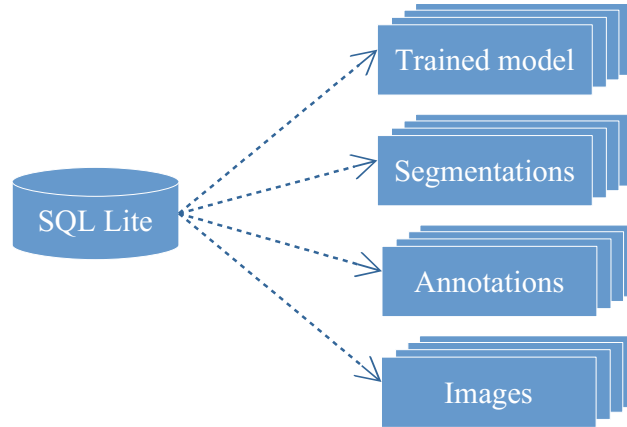


Figure 22: The central repository consisting of a SQL Lite database and a file system storage for annotations, segmentations, images, and trained model’s data.

The file system stores images, annotations, serialized model’s weights, and segmentation results. All the training images are shared among projects and maintained in their pristine state, while annotations and segmentations are project-specific and are maintained for the lifetime of the project. The SQL Lite database stores volatile information that requires synchronization such as updates from learning model and interactive input from user. This database enable us to offload synchronization issues related to reading and writing data to the SQL Lite framework.

Summary

In this chapter, we presented our approach to image segmentation by outlining the design and architecture for performing assisted labeling of images using a deep neural networks model trained with user annotations. We described the various components of the system, their user interfaces, and their corresponding input and output data. Issues pertaining to data integrity and performance were discussed along with solutions. Next, we will present an evaluation of our system under an experimentation setup to measure its performance against the conventional system of manual labeling of images by human experts.

Chapter 4 Results

In this chapter, we present the results of the project. We first describe our evaluation methodology, detailing the metric used to measure performance. Then, we present the results of evaluating the MLP and CNN algorithm in interactive and offline environments.

Evaluation Methodology

To quantitatively evaluate the performance of our system, we use Variation of Information (VI) (Meilă, 2005) to compare segmentation output generated by the system in interactive and offline modes.

Variation of Information

Kaynig et al. (2015) described VI as a metric that compares two segmentation regions S_1 and S_2 based on their entropy H and mutual information I as follows:

$$VI(S_1, S_2) = H(S_1) + H(S_2) - 2I(S_1, S_2).$$

Where the entropy H measures the randomness of the segmentation, and the mutual information I measures the information that the two segmentations share. The above equation can be re-written as:

$$VI(S_1, S_2) = H(S_1|S_2) + H(S_2|S_1).$$

Therefore, the VI can be summarized as measuring the sum of information loss and gain between the two clusterings, which indicates the extent to which one cluster can explain the other. The values generated by VI metric are non-negative, with lower values representing greater similarity.

Evaluation Data and Usage

To evaluate our models, we compare our interactive model against a model trained offline on the full data set described in chapter 2: Data Collection and Curation.

This means in offline mode we use 240 gray value images with corresponding membrane and background labels. 210 are used for training, twenty for validation, and ten for test. Note that we are only using half of the available test data to make the evaluation consistent with the interactive model. We train the classifiers by randomly drawing an equal number of samples from each image, then iteratively perform training until convergence is achieved. The trained classifier is then used to generate evaluation output.

In interactive training, we use twenty gray value images, ten from the training set to capture training annotations, and ten from the validation set to capture validation annotations. We train the classifier in the background with sparse annotations collected interactively from user. The annotation process continues until an acceptable quality of segmentation is achieved, after which the classifier is disconnected and used to generate evaluation output.

Model Setup

We prepare two sets of configuration for training CNN and MLP classifiers, depicted in Table 1. The same set is utilized for interactive as well as offline training. The MLP model consists of three fully connected hidden layers each consisting of 500 units utilizing the rec-

tified linear activation function. The size of the input layer is based on the patch size of the feature vector and is set to 39^2 units. The output layer has two units, one for membrane and the other for non-membrane.

Parameter	CNN	MLP
Learning Rate	0.01	0.01
Momentum	0.9	0.9
Patch Size	39	39
Batch Size	128	128
Hidden Units	200	500,500,500
Number of Kernels	48,48	N/A
Kernel Sizes	5,5	N/A
Classification Threshold	0.5	0.5

Table 1: Evaluation Configuration Settings

The CNN model consists of two convolutional layers, each consisting of 48 filters of size 5×5 . Each convolutional layer is followed by a max-pooling layer and the rectified linear function is used for activation. Following the convolution and max pooling layers is a fully connected layer consisting of 200 units using the rectified linear activation function. The output layer has two units, one for membrane and the other for non-membrane.

The size of the training and validation sets are different for each model. For offline, we draw a maximum of 700,000 samples per epoch, all of which are submitted to GPU for training. After each epoch we re-sample the training data again from the whole set of 210 training images. Effectively leveraging all available annotations.

Measurement Process

In the measurement process, we utilized Mahotas imaging library to generate label clusters from the probability map of the test images. Mahotas is a computer vision and image

processing library for Python (Coelho, 2012). It contains many optimized image processing algorithms implemented in C++ and operates on numpy arrays.

To generate VI, we use a set of ten test images that the trained classifiers have never seen. For each image, we generate a probability map that represents the classification output of the image. Then we threshold the probability map 1000 times with threshold values ranging from 0.01 to 1.0. For each threshold, we invert the probabilities map and use Mahotas connected components routine to label it. The labeled probability map is paired with the corresponding ground-truth labels to generate the VI results. We then compute the final VI results for each threshold as an average of the ten test images. For baseline comparison, we took a single gray value image from the training set and thresholded it following the same criteria to generate the baseline VI.

The test images used to compute the VI have an imbalanced number of pixels in each label, as such a large volume of the pixels is skewed toward one label, in this case the non-membrane label; therefore the results will reflect this imbalance.

Variation of Information Results

The following are VI results of evaluating CNN and MLP classifiers after interactive and offline training. The interactive classifiers typically converge after one hour of training time and the offline model after about 2 hours. In the following graphs, the y axes represents the variation of information, where a higher value indicate greater difference between the ground truth and the probability map clusters, while lower values indicate greater similarity. The x axes represents threshold values used to generate the variation of information. Because we're evaluating two classes of objects, in interactive mode, we used a threshold value of 0.5

for training, which is depicted in these graphs with a dotted line. For evaluation, the threshold value ranges from 0.0 to 1.0. The threshold value is a decision value that we use to segment the probability map, where probability values greater than or equal to the threshold are considered background objects and probability values less than the threshold are considered membrane objects.

Figure 23 shows the graph of VI measurements for a CNN classifier in interactive and offline modes. The interactive classifier was trained on 185990 pixels sparsely annotated on ten gray value images, i.e., 1.7% of total pixels in the ten training images. The offline classifier was trained with 210 fully annotated images. In total this is more than 1000 times the amount of data available to the interactive network.

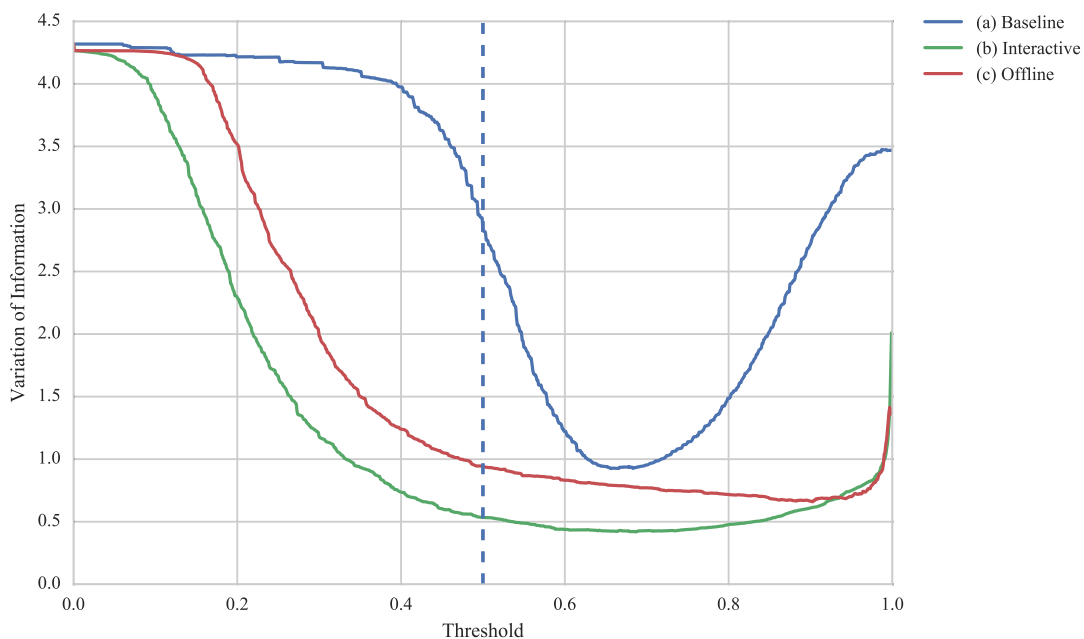


Figure 23: CNN Variation of Information: (a) In blue is baseline of gray value image. (b) In red is a CNN classifier trained on ground truth labels in offline mode. (c) In green is a CNN classifier trained on interactive annotations from user.

The blue graph at the top of the Figure 23, represents the segmentation performance of simply thresholding the initial gray value images. The red graph shown in the middle, represents the offline CNN classifier whose results outperform the baseline. The green graph shown at the bottom of the figure is the interactive classifier, which achieved the lowest VI values with the best threshold being 0.7. This means the interactive classifier is more confident of recognizing background and membranes at threshold 0.7. The results signifies the strength of our interactive CNN classifier, in that, with less than 2% sparsely annotated pixels, we can automatically label an entire image of brain circuitry and achieve better results than the conventional method of manually labeling all pixels. Therefore, our goal of reducing manual labeling of brain images is achieved.

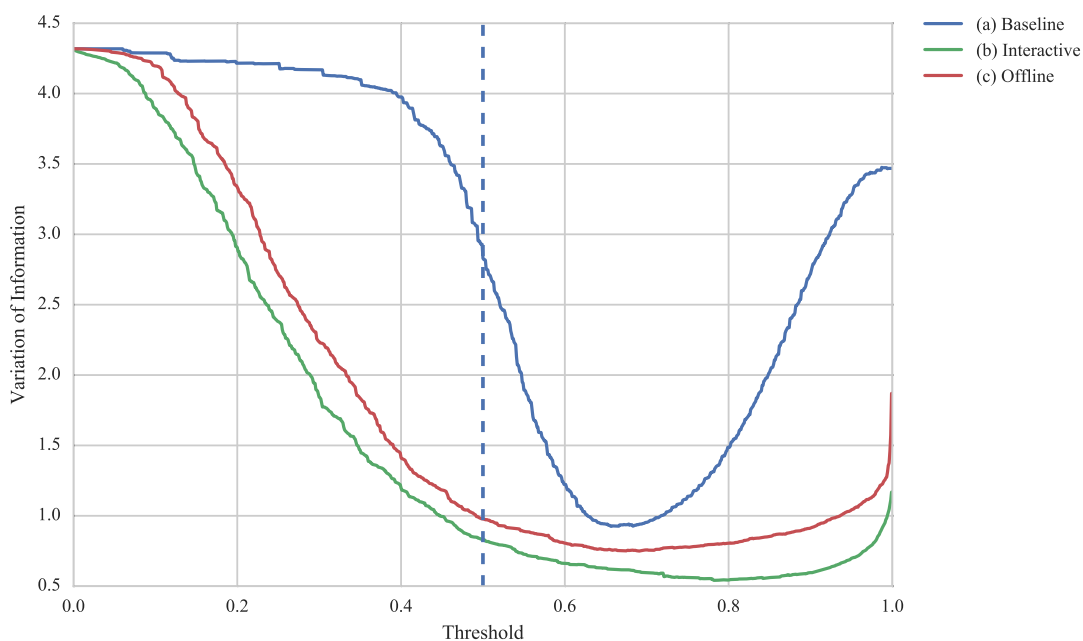


Figure 24: MLP Variation of Information: (a) In blue is baseline of gray value image. (b) In red is a MLP classifier trained on ground truth labels in offline mode. (c) In green is a MLP classifier trained on interactive annotations from user.

Figure 24, shows the results of variation of information on interactive and offline MLP classifiers. The interactive classifier was trained on 129029 pixels sparsely annotated on ten gray value images, i.e., 1.2% of total pixels. The offline classifier was trained with 210 fully annotated images. Similar to the CNN, the results shows the interactive MLP model outperforming the baseline and offline evaluation with less than 2% of pixels annotated in ten images.

Strengths and Weaknesses

The main strength of the system lies in its ability to automatically label an entire image from sparse annotations of a fraction of the image's pixels. We have demonstrated that with interactive annotation of less than 2% of an image's pixels, our system can perform automatic labeling of an image with very little human intervention. This is important for Connectomics because it significantly reduces the amount of effort required to label brain images. With this system, a fully trained model could be use to perform automatic segmentation of images in a batch mode or integrated to an external application.

The reduction of manual effort is facilitated by the distributed annotations capability, another strength of the system. With distributed annotations, we enable multiple users to simultaneously annotate images while a running model trains and guides them in the annotation process by providing segmentation feedback. The segmentation feedback allows users to correct miss-classified pixels by providing more examples to the system. This interactive feedback loop makes it possible to quickly train a classifier for a task with just a small but representative sample of the data.

Our training methodology is strongly aligned with the objective of guiding the user to

correct erroneous voxel classifications. The techniques we employed to present bad-performing examples more frequently to the system in the training process were more instrumental in improving segmentation quality than indiscriminately sampling examples. This strategy enables us to provide immediate feedback to users while training progresses.

Finally, we build the system for high-performance computing, as such, it takes advantage of GPU architecture to accelerate feature calculations and improve training time. On the CPU, we avoid long running calculations by utilizing Theano's capabilities to offload calculations to the GPU. These techniques allowed us to improve performance, as such we're able to achieve segmentation of a 1024 x 1024 image in less than 30 seconds.

An area that our system suffer from is initial model start-up. Because all DNN models created in the system start training with no prior knowledge, they suffer from a cold-start problem. Initial training of a classifier requires a minimum of 2K examples before it can produce useful feedback to user. This process sometimes takes longer in CNNs because their architecture require large amount of data. One idea to resolve this issue is to start the system with some pre-annotated data that it can learn from.

Summary

In this chapter, we presented our evaluation methodology, which consisted of computing variation of information to measure performance of the system against ground truth labels generated by human experts. We demonstrated that our interactive classifiers are able to outperform the conventional method of labeling image with less than 5% of sparsely annotated pixels. In the next chapter, we provide a summary of the thesis project and provide directions for future work.

Chapter 5 Summary and Conclusions

In this thesis, we focused on the problem of identifying neuronal structures in images of brain circuitry with the aim of developing a software framework that can be applied to segmentation problems in the domain of Neuroscience and beyond. Towards solving this problem, we designed and implemented an interactive framework for image segmentation that leverages deep neural networks hierarchical learning abilities to detect membranes and other organelles in brain images. We employed good software development techniques to build flexibility into the system so that different DNN algorithms can be substituted based on the requirements of segmentation task. Our implementation supports MLP and CNN architectures by default and can be extended to include others. We tested our framework by evaluating its performance in interactive and offline modes and produced promising results that we described in chapter 4.

Contributions

The main contribution of this thesis is a software framework that enables automatic segmentation of images from distributed interactive annotations using deep neural networks architectures. We developed iCon (Interactive Connectomics), as a multi-tiered application combining a user-friendly web-based interface with a set of state-of-the-art DNN algorithms. The approach and methodology employed is robust and works across many types of images. We primarily developed the system as a contribution to the Neuroscience community to ad-

dress the manual and tedious process of labeling neuronal structures in brain images. The resulting software, however, is applicable to segmentation tasks beyond the field of Neuroscience. The key contributions of our system includes:

Interactive DNN Training

A novel technique for training a DNN classifier that is strongly aligned with the objective of guiding a human-expert to correct voxel miss-classification. In order to achieve real-time performance and provide immediate feedback to users, we employed a stratified data sampling strategy that focuses on re-sampling data after each training epoch based on a heuristic that bad performing examples should be seen more frequently by the network in order to fast-track learning. On each epoch, we split the training batch that we submit to GPU based on the classification accuracy of the examples. We to retain up to 50% of the bad examples and re-sample the rest from the overall training data set. This strategy works very well and achieved considerable improvement in learning and segmentation feedback to user.

Distributed User Interface

A rich distributed graphical user interface based on web technologies that provides the means for capturing annotations from multiple users in parallel. We optimized the user interface to run on a web browser by leveraging HTML native capabilities to render graphics efficiently. As such, our UI is automatically cross-platform and does not require additional configuration from user.

Multi-Class Segmentation

An extensible object class system that supports the definition of arbitrary number of classes of objects to be recognized. This feature expands the application of the system to problem domains that require recognition of complex structures in images.

DNN Testbed

A flexible interface for quickly designing the best DNN network for a segmentation task. We provide advance users with the tools to configure and tune the hyper parameters of DNN networks and the ability to test the parameters with a small representative sample of the data. This facility enables users to compare DNNs performances and choose the architecture and configuration that suit their requirements.

Future Work

We designed our DNN framework to run the training and segmentation modules in parallel on separate GPUs. In training, we're able to run mini-batches in parallel on GPU; however, to fast-track the learning algorithm, we need to distribute the training across multiple compute nodes. This idea, we would like to explore in a future revision of the software.

Another aspect of the project that we would like to implement in a future revision is collaborative annotation where multiple users can annotate the same image at the same time. This is a bit challenging because of the synchronization and network latency issues involved. We do not have enough research in this area to illustrate how this can be accomplished, but one technique we'll look at is the operational transformation technique employed by Apache Wave and Google Docs. This technique replicates shared documents to local user storage so

each user can perform editing independently without a need to acquire a lock from a server. Local edits are propagated to the server and the server synchronizes all changes so that all users sees the same copy. Remote changes arriving to local copy are transformed and then executed. The lock-free and non-blocking nature of this technique makes it suitable for implementing collaborative annotations (Wikipedia, 2016). However, the issue of resolving conflicts between multiple users annotating the same area of image is not addressed by this method and require further research.

References

- Arganda-Carreras, I., Turaga, S. C., Berger, D. R., Ciresan, D., Giusti, A., Gambardella, L. M., Schmidhuber, 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., , and Seung., H. S. (2015). Crowdsourcing the creation of image segmentation algorithms for connectomics. *Frontiers in Neuroanatomy*, 9 (142):494–500.
- Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. Technical Report Arxiv report 1206.5533, Université de Montréal.
- Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: A cpu and gpu math expression compiler. *Austin, TX: Proceedings of the Python for Scientific Computing Conference*, June 30 - July 3.
- Briggman, K. L. and Bock, D. D. (2012). Volume electron microscopy for neuronal circuit reconstruction. *Current Opinion in Neurobiology*, 22:154–161.
- Ciresan, D., Giusti, A., Gambardella, L. M., and Schmidhuber, J. (2012). Deep neural networks segment neuronal membranes in electron microscopy images. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 2843–2851. Curran Associates, Inc.
- Ciresan, D. C., Meier, U., Masci, J., Gambardella, L. M., and Schmidhuber, J. (2011). Flexible, high performance convolutional neural networks for image classification. *International Joint Conference on Artificial Intelligence*, page 1237–1242.
- Coelho, L. P. (2012). Mahotas: Open source software for scriptable computer vision. *CoRR*, abs/1211.4907.
- Deng, L. and Yu, D. (2014). Deep learning: Methods and applications. *Found. Trends Signal Process.*, 7(3-4):197–387.
- Dirnberger, J. M. (2016). Biology 2108 lecture physiology: Nervous system. Retrieved February 2, 2016 from <http://science.kennesaw.edu>.
- Dory, M., Parrish, A., and Berg, B. (2012). *Introduction to Tornado*. O’Reilly Media, Inc, N Sebastopol, CA.

- Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2008). The pascal visual object classes challenge 2008 results. *Doklady AN SSSR (translated as Soviet. Math. Docl.)*, 269:543–547.
- Freeley, S. C. and Pengelley, F. J. H. (2010). Atlas of brain and injury. Retrieved January 10, 2016 from: <http://www.finr.net/files/brain/index.htm>.
- Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202.
- Glasner, D., Hu, T., Nunez-Iglesias, J., Scheffer, L., Xu, S., Hess, H., Fetter, R., Chklovskii, D., and Basri, R. (2011). High resolution segmentation of neuronal tissues from low depth-resolution em imagery. In *Proceedings of the 8th International Conference on Energy Minimization Methods in Computer Vision and Pattern Recognition, EMM-CVPR'11*, pages 261–272. Berlin, Heidelberg: Springer-Verlag.
- Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*.
- Gudi, A. (2015). Recognizing semantic features in faces using deep learning. *CoRR*, abs/1512.00743.
- Haehn, D., Beyer, J., Pfister, H., Knowles-Barley, S., Kasthuri, N., Lichtman, J., and Roberts, M. (2014). Design and evaluation of interactive proofreading tools for connectomics. *IEEE*.
- Helmstaedter, M., Briggman, K. L., and Denk, W. (2011). High-accuracy neurite reconstruction for high-throughput neuroanatomy. *Nature Neuroscience*, 14(8):1081–1088.
- Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.
- Jurrus, E., Whitaker, R. T., Jones, B. W., Marc, R., and Tasdizen, T. (2008). An optimal-path approach for neural circuit reconstruction. In *ISBI*, pages 1609–1612. IEEE.
- Kandel, E. R., Schwartz, J. H., Jessell, T. M., and Mack, S., editors (2013). *Principles of neural science*. New York, Chicago, San Francisco: McGraw-Hill Medical.
- Kaynig, V., Fuchs, T. J., and Buhmann, J. M. (2010). Neuron Geometry Extraction by Perceptual Grouping in ssTEM Images. *IEEE Computer Society*, 0:2902–2909.
- Kaynig, V., Vazquez-Reina, A., Knowles-Barley, S., Roberts, M., Jones, T. R., Kasthuri, N., Miller, E., Lichtman, J., and Pfister, H. (2015). Large-scale automatic reconstruction of

- neuronal processes from electron microscopy images. *Medical Image Analysis*, 22:77–88.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C., Bottou, L., and Weinberger, K., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Red Hook, NY: Curran Associates, Inc.
- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.
- Lucchi, A., Smith, K., Achanta, R., Lepetit, V., and Fua, P. (2010). A fully automated approach to segmentation of irregularly shaped cellular structures in EM images. In *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2010, 13th International Conference, Beijing, China, September 20-24, 2010, Proceedings, Part II*, pages 463–471.
- Meilă, M. (2005). Comparing clusterings: An axiomatic view. In *Proceedings of the 22nd International Conference on Machine Learning, ICML '05*, pages 577–584. New York, NY: ACM.
- Mishchenko, Y., Hu, T., Spacek, J., Mendenhall, J., K.M. Harris, K., and Chklovskii, D. (2010). Ultrastructural analysis of hippocampal neuropil from the connectomics perspective. *Neuron*, 67(6):1009–1020.
- Morgan, J. L. and Lichtman, J. W. (2010). Why not connectomics? *Nature Methods*, 10(6):494–500.
- Rossum, G. (1983). A method for unconstrained convex minimization problem with the rate of convergence $\mathcal{O}(1/k^2)$. *Doklady AN SSSR (translated as Soviet. Math. Docl.)*, 269:543–547.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:9.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.
- Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 3642–3649, Washington, DC, USA. IEEE Computer Society.

- Schroff, F., Criminisi, A., and Zisserman, A. (2008). Object class segmentation using random forests. In *Proceedings of the British Machine Vision Conference*, pages 54.1–54.10. BMVA Press.
- Seung, S. (2012). *Connectome: How the Brain's Wiring Makes Us Who We Are*. Geneva, IL: Houghton Mifflin Harcourt.
- Sommer, C., Straehle, C. N., Köthe, U., and Hamprecht, F. A. (2011). Ilastik: Interactive learning and segmentation toolkit. In *ISBI*, pages 230–233. IEEE.
- StatSoft (2006). Support vector machines (svm) introductory overview. Retrieved March 6, 2016 from <http://www.statsoft.com/textbook/support-vector-machines>.
- Sutton, R. S. (1986). Two problems with backpropagation and other Steepest-Descent learning procedures for networks. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum.
- Theofilatos, K., Likothanassis, S., and Karathanasopoulos, A. (2012). Modeling and trading the eur/usd exchange rate using machine learning techniques. *ETASR - Engineering, Technology and Applied Science Research*, 2:269–272.
- Venkataraju, K. U., Paiva, A. R. C., Jurrus, E., and Tasdizen, T. (2009). Automatic markup of neural cell membranes using boosted decision stumps. In *Proceedings of the Sixth IEEE International Conference on Symposium on Biomedical Imaging: From Nano to Macro*, pages 1039–1042. Piscataway, NJ: IEEE Press.
- Wan, L., Zeiler, M. D., Zhang, S., LeCun, Y., and Fergus, R. (2013). Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013*, pages 1058–1066.
- White, J. G., Southgate, E., Thomson, J. N., and Brenner, S. (1986). The structure of the nervous system of the nematode *Caenorhabditis elegans*. *Philosophical Transactions of the Royal Society of London B. Biological Sciences*, 314:1–340.
- Wikipedia (2016). Operational transformation. Retrieved March 26, 2016 from https://en.wikipedia.org/w/index.php?title=Operational_transformation&oldid=711604671.
- Yang, H.-F. and Choe, Y. (2009). 3d volume extraction of densely packed cells in em data stack by forward and backward graph cuts. *IEEE*, pages 47–52.