Deep Learning methods for 3D segmentation of neural tissue in EM images

Benjamin Eisner

An Undergraduate Thesis
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Bachelor of Science in Engineering

Recommended for Acceptance
by the Department of
Computer Science
Adviser: Sebastian Seung

May 5, 2017

Abstract

Current state-of-the-art methods in the 3D segmentation of EM stacks typically rely on a multi-stage processing of input data. Roughly, data processing consists of: acquistion, realignment, preprocessing, representation transformation, and post-processing. While most techniques strive to be fully automatic in each of these stages, errors inevitably occur. When they occur, they must be manually corrected; otherwise the errors will inevitably propogate through the rest of the pipeline to the detriment of the output segmentation. In this paper, we will explore various methods of improving accuracy at two of the stages: image transformation, and realignment. Specifically, we explore deep-learning based approaches that maximize image transformation performance on well-aligned data, and then explore various methods of learned realignment to make processing robust to misalignment. We find that while hand-crafted alignment methods currently outperform learned alignment methods, the training results suggest that further exploration of more sophisticated learned realignment schemes could potentially outperform hand-crafted methods. Additionally, we release a modular segmentation framework, DeepSeg, that allows for automatic segmentation of EM datasets and provides a flexible way to experiment with different techniques.

Acknowledgements

There are a great number of people who were either direct or indirect contributors to the creation of this thesis, and I would be remiss if I failed to mention every single one.

First and foremost, to my adviser Professor Sebasian Seung, for finding 2 hours every week to guide and mentor me and our thesis group. Your input was invaluable (as was your generous support for acquiring compute resources!). There is no way any of us could have made any headway on the segmentation problem without you. I've learned more in these two semesters of being your advisee than I have in any class at Princeton; for that I am truly grateful.

To Kisuk Lee, for knowing the answers to my questions before I even asked them, and for providing guidance to our group. I aspire to one day know as many tricks about training neural nets as you do.

To the members of the Seung Lab: Ignacio Tartavull, Dodam Ih, Jonathan Zung, and William Wong. Thank you for your inexhaustable willingess to help our group understand the subproblems in the segmentation task, and for being available at altogether unreasonable hours to answer our Slack questions.

To the non-Frank members of my thesis group: Sharon You, Evelyn Ding, and Nathan Lam. You picked up my slack when I had unproductive weeks, you helped me debug my learning problems, you made me laugh when our work was overwhelming, you put up with (and helped me combat) my compulsion to work on infrastructure rather than actual research. Without you all, there would be no thesis. I'm so glad we grew so close this year.

To Frank Jiang, my roommate, my thesis groupmate, and one of my very best friends. We have been through so much together these last 4 years since the Bloomberg 3rd floor lounge: we've roomed together at Princeton every year since Sophomore year, lived in Lisel's windowless room together in Park Slope, studied abroad in London together, traveled Europe together, and generally found that our education/career goals had converged completely. You're one of the smartest people I know, and always willing to drop whatever you're doing to work on a hard problem. I don't know what I'm going to do without you next year, bud.

To Benjamin "Careful Danger" Leizman, for your endless support and bottomless friendship. We've been around the block together, and you've always been there for me when I needed you. You're a class act, and I love ya, man.

To Keith Gladstone for inspiring me on a daily basis with your hopeless, hopeless romanticism.

To the group message formerly known as edges2cats (Elias Rubin, Vlad Feinberg, Thomas Hartke, and Keith), for wasting my time with pointless conversations that somehow I always thoroughly enjoyed.

To the Cloister Lifeguards, for being one of the most consistent, cohesive groups of people I've ever had the pleasure to call my friends.

To my T8 Fam and my Sympeeps, for being the most fun I've had at Princeton, hands down. I would have snapped my laptop in two if I didn't have dance this year.

To my second reader, Professor Thomas Funkhouser, for taking the time to read 70+ pages about this thing I did.

To Sebastian Riedel, Tim Rocktschel, and Isabelle Augenstein for exposing me to learning with TensorFlow long before I started this project.

To the Princeton Office of Stewardship and the Princeton Office of Financial Aid: I certainly would not be here without the extremely generous financial aid grants I've received.

And, last but not least, to my parents. Mom and Dad, you taught me everything I know. You set me up for success at every turn, and always provided me so much love and support. I love you both immensely. I hope I've made you proud.

To my parents.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Reconstructing the human connectome at the neuron-level is a daunting task. It would take a trained neuroscientist roughly 400 trillion hours to manually reconstruct the 3D-geometry of an entire human brain from cell-level electron microscopy images of brain tissue[1]. Considering that the universe has only existed for roughly 112 trillion hours, it is unlikely that humans will ever manually reconstruct the entire human connectome. And yet, knowing the entire neuron-level human connectome would be eminently useful across the field of neuroscience. We can do better - not with humans, but with machines.

[1]This is a rough lower-bound estimation, assuming that that it takes a neuroscientist roughly one hour to reconstruct the geometry of a $6\mu m \times 6\mu m \times 200nm$ section of tissue and that the average human brain has a volume of $1300\mathsf{cm}^3$.

1.1 Overview of Contributions

This thesis is an exploration into various automated methods of reconstructing the 3D geometry of neural tissue through image segmentation. Specifically, we attempt to increase the performance of existing automatic EM segmentation pipelines, both in efficiency and accuracy, by exploring modifications at various stages of these pipelines.

Throughout our initial exploration of existing EM segmentation methods, it became increasingly clear that, since the output of one stage of the segmentation pipeline feeds directly into the next, errors at any given stage will inevitably propagate to later stages. There are generally two approaches to mitigating this propogation effect: reduce the errors introduced at any given stage (i.e. increase the accuracy of an intermittent neural net), or make subsequent stages more robust to errors in previous stages (i.e. add substantial augmentations to training, apply techniques like Mean Affinity Agglomeration to segmentations). This thesis touches on both categories of improvement.

While this thesis research was conducted in conjunction with several other undergraduates, graduate students, and a professor in the Princeton Neuroscience Institute, this thesis will detail my individual contribution. Specifically, my contribution can be broken up into three parts:

- The creation of the `DeepSeg` segmentation pipeline, a modular framework written in Python that allows for easy training and prediction with current popular models, as well as easy experimentation at different stages in the computational pipeline.

- Experimention with several different architectures for transforming raw segmentation images into affinity/boundary maps, attempting to improve overall accuracy and noting their invariance to errors in earlier stages of the pipeline.
- Exploration of learned alignment strategies, both on learned transformations and in an end-to-end setting.

1.2 Motivation

Understanding the cellular structure and connectivity of neural tissue is perhaps the most important challenge in the field of computational neuroscience today. The accurate generation of a complete neuron connectivity graph of a section of neural tissue would allow researchers to answer a litany of fundamental questions about neural activity at various scales. At the micrometer level, being able to generate connectivity graphs gives us insight into how dense connections are throughout the brain, answers quesions about local clustering and distribution of synapses, and shows us how loosely or strongly coupled neurons can be.

At the millimeter scale, being able to reconstruct the topology and connectivity in a small region of brain tissue would allow us to understand how entire neurons interact with each other, and how clusters of neurons interact. This scale would allow for queries into specific neural subsystems, especially in very small animals (i.e. insects, small rodents), and perhaps allow us to understand the types of stimuli that affect entire neurons.

At the centimeter scale, we can map the entire brain of small creatures, and important structures within the human brain. A full map at this level would allow us to query memory structures in small animals, simulate reactions to external stimuli in the optical system, and examine the differences in neural structures between different individuals (at smaller scales, it is difficult to find isomorphic sections of tissue).

Finally, mapping neural tissue at the scale of the human brain would allow us to begin to be able to answer fundamental quesions about memory, consciousness, and humanity. Given enough computational power, it might even be possible to simulate a specific consciousness. This is quite a lofty goal, especially considering the computational requirements, but advances in automated methods make these tasks more and more plausible.

Conventionally, the structure of the brain is inferred from images, whether they are thin slices of a brain imaged with an electron microscope, volumetric images acquired using digital radiograhpy systems (i.e. fMRI, CAT, etc), or visible-spectrum video of exposed brain tissue. Although these imaging technique generate information at different resolution levels, they invariably present a huge data problem: when researchers are presented with small-scale image data, it is fundamentally infeasible to efficiently infer the connectivity and structure of a small cluster of neurons by hand, let alone an entire brain or nervous system, simply because the amount of neurons in a brain is too large. For several decades now, researchers have had the capability to manually reconstruct the 3D geometry of tissue in tissue volumes at the micrometer scale[24]. However, to get to larger scales within acceptible time frames, more automated methods are necessary.

Many attempts have been made to automate the process of inferring connectivity and topology from images using various algorithmic and machine learning models. In the past five years or so, many of the most successful attempts at this class of problems have utilized Convolutional Neural Networks (CNNs) to achieve their high performance. The goal of this year-long project is to explore many of the different CNN-based approaches that have gained recognition in the past few years in several sub-problems, evaluate their performance and enumerate their deficiencies, and attempt to design new architectures that achieve improved performance in these sub-problems. In addition to increasing performance on established benchmarks, we also make contributions on new sub-problems for which there are no established benchmarks.

The motivation for the research in this field is to better understand the connectivity of neural tissue. Since this is such a broad goal, it stands to reason that there are a number of intermediary sub-problems that can be tackled to learn about connectivity. Several of the sub-problems have been heavily studied, and various public competitions have been organized that provide labeled training data and unlabled test data, encouraging competitors to achieve maximum performance against a certain benchmark. As we developed our models, we evaluated their performance on local train sets and submitted their predictions to several of these open competitions, often performing well.

1.3 Related Work

1.3.1 Connectomics

The problem of determining the connectivity of a brain falls in the sub-field of connectomics, which has been a lively area for research for over 30 years. The first full connectome of an organism was created in 1986, producing the mapping of the brain of *C. elegans* [24]. Since then, partial and full topological and connectivity maps have been created on various organism, often using electron microscopy and careful hand-reconstruction to do so. One group of researchers from the Allen Institute for Brain Science was able to use the presence of flourescent proteins to construct a cellular-level connectome of a mouse brain (although not at the level of accuracy to reconstruct a true weighted graph of the connectome)[21]. Other researchers have been able to automatically reconstruct certain cell structures at nano-resolution[14]. More recently, by genetically modifying organisms to produce proteins that become phosphorescent in the presence of calcium (calcium is released across the synapse between a dendrite and an axon when a neuron is fires), researchers have been able to monitor both brain activity and neural structure using video photography in the visible spectrum [20].

1.3.2 Image Segmentation

As the field of connectomics has matured, it seems to have coalesced around the idea that taking electron microscopy images of neural tissue and then performing image segmentation on those images is likely the most promising strategy for acquiring

high-resolution neuron graphs. Image segmentation has a long history in the field of computer vision; being able to separate various components of an image has broad applications, and much research been done on the subject. In the early 1980's, state-of-the-art methods of image segmentation techniques involved defining hand-crafted features for local structures within images in order to perform segmentation[9]. In the late 1990's and early 2000's, methods began to examine more global image features. For instance, Malik and Shi demonstrated that the segmentation problem could be formulated as a graph partitioning problem[13].

A huge breakthough in pixel-level image segmentation occured in 2015, when Long et. al. showed that fully convolutional networks could be used to achieve state-of-the-art semantic segmentation (where semantic segmentation means labeling pixels based on the object that they represent)[16]. Since then, many groups have improved on various techniques for various semantic segmentation domains[4].

1.3.3 EM Segmentation

Because of the acute interest in using EM segmentations to reconstruct neural tissue (and a broader interest in segmenting biological images), many groups have successfully applied segmentation techniques catered specifically to this task. In the early 2010's, a group of researchers published an open dataset and created a global challenge to use machine learning methods to label neurons in 2D image slices of a brain, resulting in the creation of models with near-human accuracy [3]. The first successful convolution-based model to perform well on this task was the N4 architecture, a fully-convolutional approach to segmentation released in 2012[7]. Many subse-

quent approaches have drawn on the success of N4's fully-convolutional architecture, including VD2D, and more intricate architectures like Multicut and U-Net[5, 22].

In the 3D Segmentation domain, significant progress has been made following the ISBI 2013 SNEMI3D challenge[2], which challenged entrants to segment 3D EM stacks. Prior to this, it was shown that using convolutional nets to predict 3D affinities could lead to accurate segmentation[23]. The aforementioned convolutional models have all been adapted and extended into the 3D domain, with architectures including VD2D-3D[15], 3D U-Net[6], and V-Net[19]. Other challenges with different types of data, including CREMI 2016, have also spurred model development[8].

1.4 EM Segmentation Pipeline

So far, we have referenced "EM Segmentation Pipeline" as a process that converts raw EM images into 3-dimensional segmentations of the structures those images represent. There are several computational stages of this pipeline, and in order to understand how altering the pipeline will affect overall performance it is necessary to explain in detail the various components of this pipeline. While different segmentation techniques may use some subset or superset, the pipeline I outline below is a general conceptual representation of what most state-of-the-art segmentation schemes utilize.

The EM Segmentation Pipeline can roughly be separated into five components, shown visually in Figure 1.1:

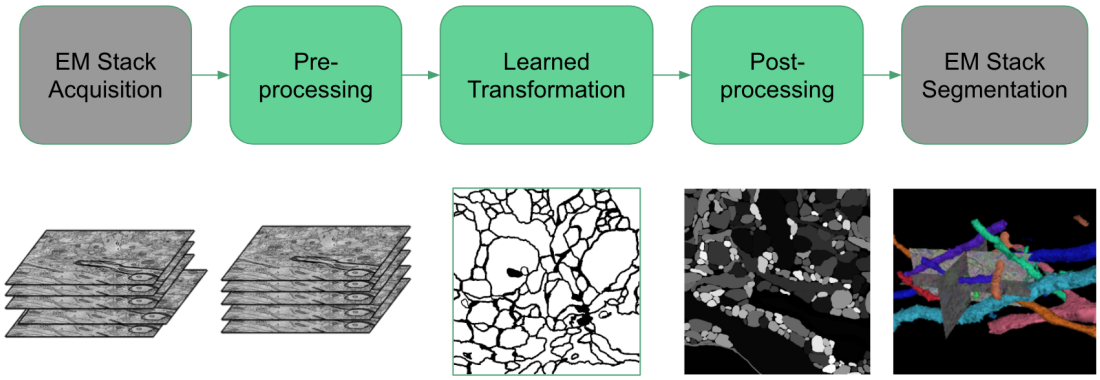

Figure 1.1: A general outline of the EM segmentation pipeline. The lower set of images represent intermediate stages that the data takes on during processing. In this example, the images are first acquired through some sort of electron microscopy technique (typically ssTEM). Second, they undergo preprocessing, which is primarily realignment of slices that were disrupted in the imaging process. Third, a learned transform is applied, which in this case transforms the stack of images into an affinity map. Fourth, postprocessing is applied, which in this stage is computing an actual segmentation from the predicted affinities. Fifth, geometric segmentation is inferred from the pixel segmentation, and the data is ready for use in a downstream task.

1.4.1 Image Acquisition

Given a physical volume of neural tissue, the first task in inferring tissue structure is to acquire some sort of digital representation of this tissue. While there are many techniques available for imaging biological tissue (e.g. light microscopy, electron microscopy, radiography, magnetic resonance imaging), typically the only way to acquire a representation of cell-level structures is by using a tunnelling electron microscope (TEM)[18]. Before imaging, samples undergo considerable preparation: typically, they are embedded in a rigid medium that will allow them to be sliced with minimal distortion; additionally some sort of stain is applied to the tissue that affects the electrical properties of different biological structures, allowing for high-contrast

imaging. The result is a set of slices of neural tissue, 30-50nm thick, that can be independently imaged in the TEM. When positional order from slicing is combined with the raw image data (which takes the form of grey-scale images, as opposed to RGB or CMYK images), each individual value can be treated as a voxel, since it is indexed by three orthogonal coordinates. Important to note is that these voxels are anisotropic, meaning that they are larger in the z-dimension than they are in the x-y dimension. This introduces a computational complexity that can somewhat be compensated for at later stages of the pipeline.

Although the actual performance of this physical imaging process is far ouside the scope of this thesis, we mention the physical steps involved because the methods used in preparing and imaging biological slices have immediate consequences on the quality of the data that is fed into stages of the pipeline in which we are primarily interested. Because the imaging process is physical and involves structures at nano-scale, physical preparation of the sample can introduce various defects into the slices that show up in resulting images. The staining process, for instance, can inconsistently vary contrast throughout an image, and can produce large dark blotches in an image. The slicing procedure can create tears and folds in tissue, which manifest as discontinuities in the images, and can even physically translate slices hundreds of nanometers. And since neural tissue naturally contains significant quantities of water, samples are prone to dry inconsistently, resulting in elastic warping of cell-level structures (like a rubber sheet that has local stretching). Visual examples of some of these artifacts can be found in Figure 1.2.

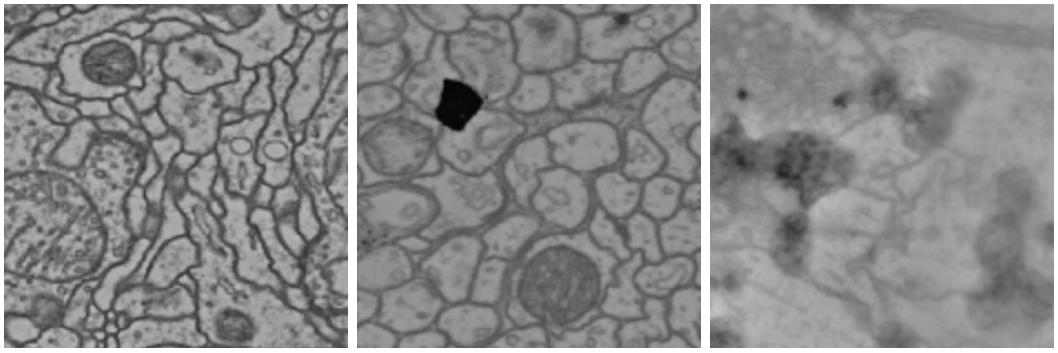

Figure 1.2: Examples of defects in the imaging process. Left: a properly stained and imaged segment. Center: a slice where inconsistent staining or another artifact has left a large dark spot on the image. Right: a slice that was prepared in such a way that the microscope couldn't produce a sharp image, either during slicing or focusing.

These deformations caused by the imaging process can have drastic implications in the performance of later stages of the pipeline, especially when those stages aren't explicitly corrected for. We noticed that even slight misalignments in image data caused ultimate segmentation performance to noticably suffer. Later in the pipeline we will explore various techniques that can be used to make the pipeline more robust to these inevitable imaging defects.

1.4.2 Preprocessing

Once volume data is acquired, data usually will undergo any of several preprocessing techniques to prepare it for later stages of the pipeline. The scope of preprocessing and types of data transformations performed vary depending on the robustness of later stages of the pipeline, as well as requirements on the output of the segmentation pipeline. Preprocessing is generally treated as distinct from subsequent stages of

the pipeline because the transformations often keep the data in the same domain of values, and maintain the data representation. In other words, both the input and the output of preprocessing take the form of stacked EM images. Typically, preprocessing will include some form of image adjustment and stack realignment.

Image adjustment can be as simple as altering the contrast on individual images, or making contrast uniform across the entire stack. While these adjustments will typically improve segmentation results, most modern deep learning techniques (which are liberally used in the subsequent pipeline stage) are easily trained to be quite robust with respect to level differences between images, so a rigorous exploration of image adjustment techniques would likely yield marginal gains in accuracy.[2]

The same cannot be said for image alignment. The defects and imprecisions introduced in the actual imaging process can severly impact segmentation performance, particularly because they introduce three-dimensional discontinuities that make it difficult for many neural networks to trace continuous segments across slices. Thus, automatic stack alignment is an active area of research.

At its heart, the alignment problem is one of misrepresentative data. Ideally we would like each 'voxel' to spatially correspond to a true volume in the sample, and for a voxel's position in the the data to correspond to its true position within the greater sample. The defects in the imaging process taint this mapping, and we are left with a dataset that, when taken literally, misrepresents the physical volume from which it was derived. Thus, the task of realignment is to take this noisy data and distort it in some way so that it more accurately corresponds to the original volume.

[2]Training speed, however, could potentially see significant improvements, as a neural net would have to learn fewer functions if its input were more uniform

While there are many theoretical ways of registering two images (registration here means alignment and distortion so that their features align accurately), most modern methods rely on establishing points of correspondence between two or more images, and distoring the images such that those points of correspondence end up at the same x-y coordinate in all images. It is generally believed that, given a high enough density of true correspondences throughout a stack, one can transform the data into a form that is pixel-for-pixel accurate with respect to the actual cellular structure of a sample.[3] The transformations themselves can be parameterized as elastic transforms, which provide discrete interpolation for all voxels not labeled as correspondences.

The problem, then, lies in actually determining these correspondences with high accuracy and high enough density for sample-accurate registration. One popular tool for achieving rough correspondences is `TrakEM2`, which provides functionality for registering generic images using a combination of the Scale Invariant Feature Transform (SIFT) and a global optimization [17]. This algorithm uses no learned or otherwise domain-specific knowledge, and is widely used across computer vision applications to stitch arbitrary images together. This process is sufficient for establishing rough correspondences, but the noisy and varied nature of cellular structures means that, without any more domain-specific correspondence labeling the resulting image registration on a moderately distorted dataset will likely not result in transformations that are smooth or accurate.

[3]Intuitively, this makes sense, since structures within cells are physically connected, our notion of correspondence is essentially a description of how these structures are physically connected.

The Seung Lab's current realignment techniques attempt to use domain-specific techniques to achieve correspondence. These techniques are typically hand-designed filters that draw on domain knowledge of the consitution of EM slices of neural tissue, and require a non-trivial amount of hand-tuning when applied. To compute a realignment, first a sparse set of correspondences are made at the macro level, and a rough realignment is iterateively computed (the rationale being that iteratively computing many fine realignments has slow convergence and is computationally infeasable). Following this rough realignment, a much more granular set of correspondences are computed, and the stack is iteratively deformed a small number of times to compute a smooth, converged registration. This technique is quite effective, and leads to near-perfect registration, but requires a substantial amount of trained human input to determine both the parameters for the various correspondence filters and to correct obviously incorrect correspondences.

A more desirable approach would be to use machine learning to train a large set of filters (more specific ones than humans could compute) to predict correspondence at different levels of granularity. This is an active area of research within the Seung Lab. Alternatively, machine learning could be used to learn the actual parameters for a piecewise affine or elastic transform, rather than learn filters to predict corresponence, although as we demonstrate in Chapter 5 this is potentially quite difficult.

Again, it is worth noting that failures in preprocessing to compensate for errors in the imaging stage - as well as new artifacts introduced in the preprocessing stage - will be propagated through the remainder of the pipeline, and can only have a negative or neutral impact on segmentation performance.

1.4.3 Image Transformation

The image transformation stage of the segmentation pipeline can loosely be defined as any set of transformations that compute a representation of the sample that is different in kind from a set of aligned images. In the case of most of the models discussed in this paper, the image transformation stage converts a stack of EM images into a 3D pixel-wise affinity map, where the labels at each pixel represent the probability that that pixel is in the same cellular body as the adjacent pixel in the x, y, and z direction. For other segmentation schemes, like Google's Flood-Filling architecture, the transformation output is a direct segmentation.

This stage is perhaps the most well-explored in the pipeline for neural segmentation. While in the past hand-crafted or generic techniques were used in this stage to some degree of success (i.e. selective thresholding, hand-crafted filters, etc), most state-of-the-art techniques utilize some sort of machine learning scheme to learn the output representation, usually a neural net architecture that utilizes many successive convolutions to predict each pixel (or a small patch) of the output segmentation. These pixel/patch predictions can be stitched together to make a prediction on a whole stack. These ML approaches are typically supervised, and rely on using large datasets annotated with segmentations for training. Thus, as in most deep learning applications, the size and diversity of the training set is one of the most important factors in maximizing the generalization performance of this stage of the pipeline.

Because large, high-quality, labeled EM datasets are hard to come by (see the second sentence of this thesis), most researchers will take a reasonably-sized dataset and randomly apply data augmentations that make the data look like fresh data,

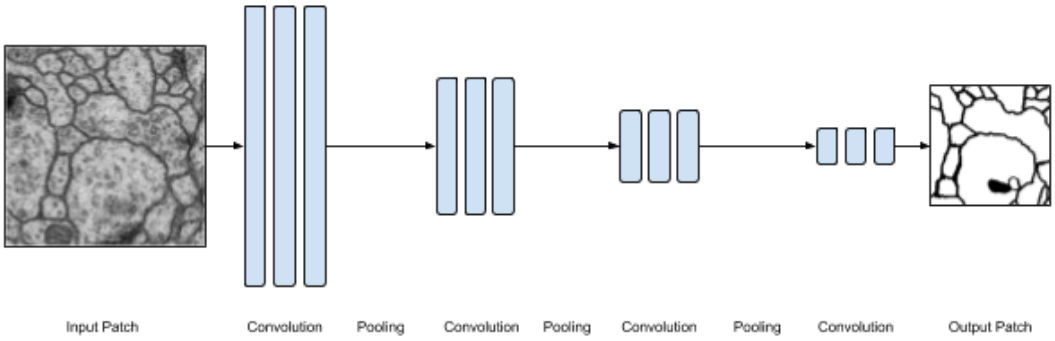

Figure 1.3: A prototypical Fully Convolutional Network, with successive convolutional layers followed by pooling layers. Depending on the types of convolution layers, the size of the input may be much larger than the size of the output, implying that the field of view for each pixel in the output could be greater than 1.

thus artificially expanding the size of the dataset. These augmentation techniqes can broadly categorized as either affine transforms or elastic transforms. Specifically, researcher will usually introduce some sort of scaling, rotation, shearing, and elastic warping to both the image stack and the labels. These techniques have an enormous impact on accuracy and generalization - empirical evidence of this is provided in Chapter 3.

The specific network architectures that are used in this stage of the pipeline are quite varied. We will describe the general classes of architectures relevant to our exploration here, and will revisit specific architectures we used later in this paper. Additionally, it is worth noting that elements of various architectures listed below can be combined (i.e. adding residual connections to standard convolutional nets.)

Standard Convolutional Networks

The most conceptually simple type of network that is used is the standard convolutional network, depicted in Figure 1.3. Standard convolutional nets typically involve several successive convolutions with small filters (e.g 3x3x3 filters) that perform either 'valid' or 'same' convolutions, which are differentiated by the size of the output of the convolution. After each convolution, a nonlinearity (e.g. ReLU) is applied, and every so often pooling layers are interspersed to alter the scale of the underlying feature maps. At the end of these alternating convolutions and poolings is a prediction, often the output of a sigmoid, that predicts a single pixel or patch of boundaries (or affinities, in the 3D case). Depending on the number of convolutions and poolings, the output pixel is determined by pixels within a certain field of view in the input image - that is, if one were to mathematically unroll the convolutions and poolings, only a certain number of pixels in the input image would be used to compute the prediction in the output image. In the N4 architecture applied in 2 dimensions, for instance, each pixel in the output prediction is influenced by pixels in a bounding box of 96x96 pixels. Architectures that are primarily Standard Convolutional Networks include N4 (2-dimensional), VD2D (2-dimensional), and VD2D-3D (3-dimensional)[7, 15].

U-Net

U-Net style architectures closely resemble autoencoders, in that they compress a representation of an input using convolutions and poolings, and then decompress that representation using convolutions and up-convolutions. However, instead of trying

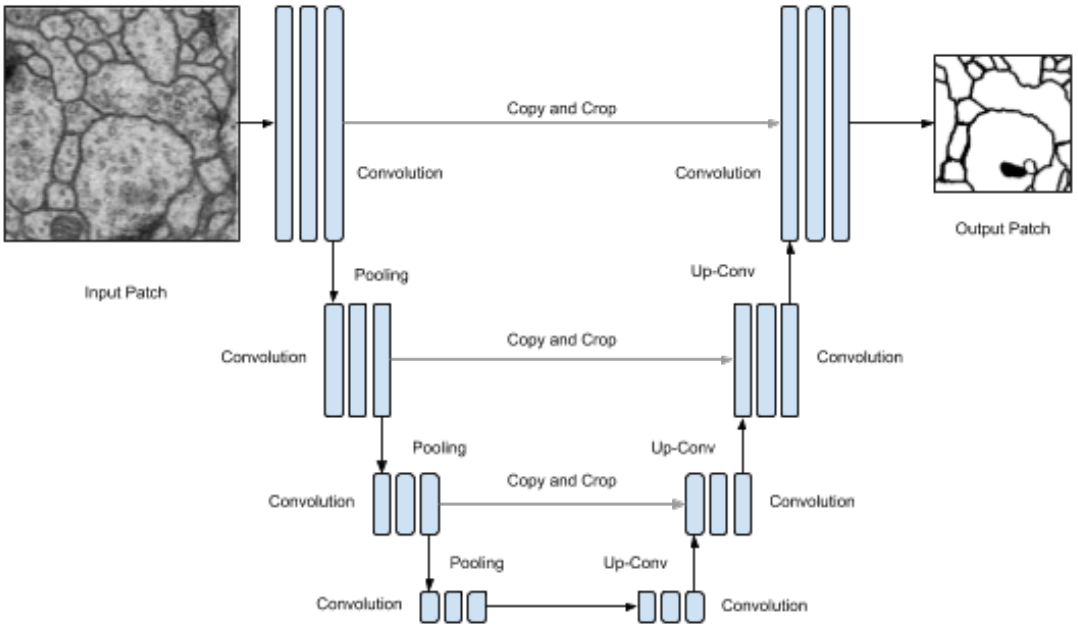

Figure 1.4: A prototypical U-Net, with successive convolutional layers followed by pooling layers on the downward pass, and then successive convolutional layers followed by up-convolutions on the upward pass. Notice how the inputs to each convolution after an up-convolution also takes as input the output of the last convolution of the corresponding layer in the downward pass.

to predict a perfect reconstruction of the input, the U-Net architeture attempts to reconstruct boundaries/affinities for the input image. One key feature of U-Nets is the symetrical use of skip connections - for every convolutional layer on the compressive (downward) pass, the output of those convolutions is added to the input of corresponding convolutional layers on the decompressive (upward) pass. This serves to force the net to quickly learn a set of affinities that look like the input. Much like Standard Convolutional Networks, every pixel in the output patch of a U-Net architecture can be affected by pixels in a certain field of view in the original input -

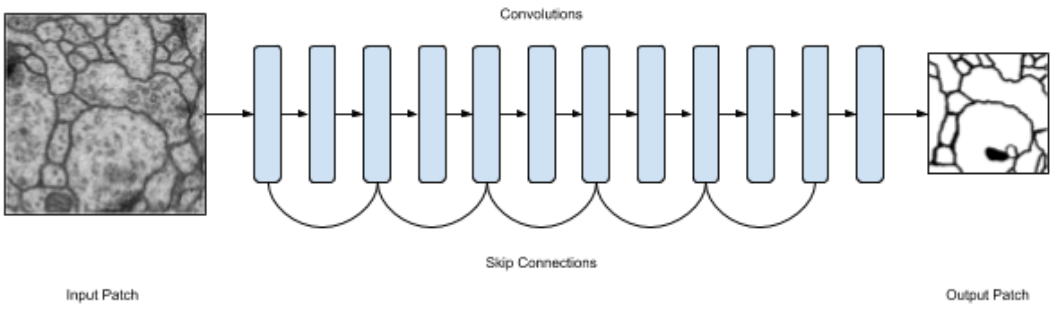

Figure 1.5: A prototypical Residual Net, which has a similar structure to the Fully Convolutional Network. The major difference is the skip connections - the residual connections - are added right before the nonlinearity but after a convolution. Residual connections can be added to other architectures to achieve some of the properties of this network.

modifying this field of view can often modify the quality of the results. Empirically, the predictions of U-Net architectures tend to be significantly smoother than fully convolutional architectures (although they may not be more accurate). A prototypical U-Net can be seen in Figure 1.4. Architectures that are primarily based on the U-Net style are the original U-Net (2 dimensional), 3D U-Net (3 dimensional), and V-Net. SegNet also resembles the U-Net architecture, in drawing inspiration from the autoencoder paradigm[22, 6, 19, 4].

Residual Nets

Residual Networks draw on a similar idea as U-Net, in that convolutional layers should be able to directly influence layers that do not directly follow them in the architecture[10]. As shown in Figure 1.5, Residual Nets add skip connections that jump over one (or more) sets of convolutions. Deeper nets are, in general, quite

difficult to train, and the use of residual connections often increases the speed of training for deeper architectures.

Flood-Filling Networks

Flood-Filling Networks (FFNs) take a different approach to the prediction task. Rather than predict whether a pixel in the output is a boundary or not on a given input, FFNs are given an input patch and an x-y coordinate, and output a pixel mask for that patch that represents whether a pixel in the output is in the same body as the input x-y coordinate[12]. If a particular object is sampled such that the output patches completely cover the object, a full segmentation of that object can be achieved (successive sample patches are chosen by a recurrent neural net (RNN)). Typically, this map is predicted with several convolutions, although architectures are varied. While we will not explore FFNs any further, as they are outside the scope of the research presented in this paper, they are noteworthy in that directly predict segmentation, rather than an intermediate (and thus require substantially less postprocessing than other menthods).

Typical Errors

The errors made by neural networks in the Image Transformation stage (and ultimately affect the accuracy of the resulting segmentation) tend to fall into several categories: split errors, merge errors, and incorrect object shaping. These errors are somewhat self-explanatory. Split errors occur when networks predict intermediates that, when converted into a segmentation, incorrectly split a body into two distinct

segments when it should be one. Similarly, merge errors occur when two distinct objects are labeled as the same object. Incorrect object shaping occurs when the output of the network incorrectly predicts boundaries that may not necessarily cause merge or split errors, but simply predict boundaries that are too thick or otherwise encroach on the true shape of the object. These sorts of errors can be mitigated either by altering network architecture, or through postprocessing.

Important to note is the fact that the quality of the input data has a huge effect on the quality of the output data. While data augmentations during training can help networks compensate for misaligned data, generally inputs that are well-aligned will result in significantly better predictions than those that are poorly aligned. It is difficult to completely correct for this at the Image Transformation stage, which is why we stress the importance of the Preprocessing stage in preparing data well for segmentation.

1.4.4 Postprocessing

Once the EM stack data has been transformed into an intermediate form (i.e. boundaries/affinities, or segementation in the case of Flood-Filling Networks), postprocessing is often applied. Postprocessing can take on many forms, but often involve using the intermediate form to prepare a segmentation. Given a boundary/affinity map, there are many different ways to infer a segmentation of an image. However, we will discuss two methods that are used in the Seung Lab to process boundaries/affinities into segmentations.

Watershed Transform

The Watershed Transform is a standard computer vision algorithm for segmenting images. Given an affinity graph, the algorithm treats the values of affinities as energy values, which is analogous to the height of land in a geographical landscape. Since pixels that belong to the same body have high affinity, the topological high-points will be the bodies themselves, and the valleys will be the boundaries between bodies. The watershed variant used in the Seung Lab identifies the plateaus (representing distinct objects) and uses those identified plateaus to inform where to place basins for the classical watershed algorithm. The algorithm then floods basins based on a set of supplied parameters. This allows a distinct labeling of all pixels in the image, where pixels in the same basin are given the same label. All bodies that are too small to actually be distinct bodies are merged greedily. It is at this stage where merge/split errors often manifest, since two basins could be split or merged if there are discontinuities in an affinity boundary, or if non-cell-wall affinites are too high. Selecting appropriate parameters for this task is paramount in finding a good segmentation, and is typically performed empirically for a given dataset.

Mean Affinity Agglomeration

One way to mitigate some of the merge/split errors that arise from the Watershed Transform's intolerence of slightly imprecise affinities is to artificially heal the imperfections in boundaries that are generally correct. Mean Affinity Agglomeration (MAA) is one way to do this. MAA iterates over boundaries between segmented objects and greedily merges or splits them based on the mean affinity along the

boundary. In this way, inconsistent boundaries predicted by the Image Transform stage can be healed.

1.4.5 Segmentation/Downstream Processing

Once postprocessing has occurred, the data is now in a completely segmented form. The segmentation can then be used for downstream tasks. Theoretical applications of segmentation include labeling different cells by their type, generating weighted directed graphs that represent connections between neurons, and determining characteristics of neurons in different types of tissue.

Chapter 2

The DeepSeg Framework

So far we have discussed the theoretical computational concepts that underpin the EM segmentation pipeline, and realized that there are many nuanced computational steps that flow into one another during the segmentation of an EM volume. Given the algorithmic intricacy of the computational tasks, it stands to reason that any software implementation of the computational tasks will be large and complex. Conventional wisdom holds that a large and complicated software package is anathema to a researcher attempting to experiment with, augment, and improve upon the techniques implemented in that software package. Repeatability of experiments is also critical to a researcher retaining his or her sanity, so a pipeline that is completely automatic and is completely specified by a consolidated set of parameters is critical.

As our research group started building and modifying software to experiment with different stages of the segmentation pipeline, it became painfully obvious that as our codebase grew in size and complexity, it was becoming increasingly difficult

to alter the software without making major modifications across the codebase. So, in order to avert collective anguish we decided to design a framework - which we tentatively have named `DeepSeg` - with the following set of goals:

- to create a set of abstractions and interfaces that allow researchers to modify or swap-out different components with minimal software-level impacts on the functionality of other portions of the pipeline.
- to define abstractions for model development that are succinct and use domain-knowledge about the problem to automatically connect to sampling mechanisms during the training process.
- to be completely portable accross different machines and host environments, and to automatically integrate with installed GPU hardware for training acceleration.

On the technical level, `DeepSeg` is written in `Python` (with some `Julia` and `C++` tools), and uses the `NumPy` package to represent and manipulate data throughout the pipeline. All of the machine-learning components, particularly for model definition and training, are built on top of the `TensorFlow`, Google's popular, open-source deep learning framework[1]. This was chosen because of its ease of use with `Python`, its flexibility, and its automatic integration with GPUs via `CUDA`.

In the following subsections, we will describe the abstractions introduced in the `DeepSeg` framework, and some of the underlying implementation details of these abstractions.

Interested readers can explore the `DeepSeg` codebase at the following URL: `https://github.com/tartavull/trace/tree/beisner`. Currently the framework lives in the "beisner" branch, but it will likely be merged with "master" soon.

2.1 Overview

The driving concept behind the design of the system is being able to specify the entire pipeline in one place. This includes specifying all dataset handling, preprocessing, image transformation, and postprocessing procedures and parameters. Additionally, for any components that require learning or optimization, training parameters and inference specification are explicitly required. Because every non-parameter component in the specified pipeline must adhere to specific interfaces (i.e. dataset samplers must provide data samples of a specific size), components can be freely swapped out in the specification with the knowledge not only that the pipeline will execute, but the only meaningful difference in execution will occur at the altered component.

In terms of functionality, the framework provides both training and inference for a specified pipeline. The training process automatically hooks into `TensorBoard`, the `TensorFlow` training visualization tool, in order to monitor training progress. The pipeline can automatically load trained models for inference tasks, and supports exporting into formats accepted by various EM segmentation competitions.

2.2 Pipeline Specification

A pipeline can be completely specified with a set of parameter classes:

- `PipelineConfig`: The main configuration class, which contains all other sets of parameters, as well as which models are used, where to find datafiles, and where to save results.
- `TrainingParams`: The set of parameters used in a training process, including learning rate, optimizer, patch sample sizes, and batch sizes.
- `AugmentationConfig`: A set of booleans determining which augmentations to use when sampling the dataset.
- `InferenceParams`: The set of parameters used when performing inference, including how to assemble predictions on large images from smaller predictions.

A `PipelineConfig` object is passed to the `Learner` class, where all the relevant componenets are connected.

2.3 Handling Diverse Datasets and Label Types

Currently, the framework supports several different datasets out of the box: the ISBI 2012 EM boundary-detection dataset, the ISBI 2013 SNEMI3D EM segmentation dataset, and all the datasets provided by the CREMI 2016 EM segmentation challenge. Prediction preparation tools are available to reformat the predictions on test sets for submission to their respective leaderboards. The framework also supports arbitrary EM datasets of any reasonable size,[1] and supports label inputs as segmentations, boundaries, 2D affinities, and 3D affinities.

[1] Any dataset that fits in RAM.

Raw datasets are wrapped by classes that implement the `Dataset` interface (i.e. `CREMIDataset`, `SNEMI3DDataset`, and `ISBIDataset`).

2.4 Dataset Sampling

The framework provides several different modes for sampling a specified dataset during training or inference. For training, random samples of arbitrary shape can be sampled, to which specified augmentations are applied. For inference, entire validation and test sets can be sampled in formats that are appropration for feeding into the pipeline.

2.4.1 Augmentation

The framework supports several type of random augmentation:

- Rotation: the entire stack can be rotated by a random angle.
- Flipping: the entire stack can be randomly mirrored along the x, y, or z axis.
- Blurring: individual slices within a stack can be arbitrarily blurred.
- Warping: individual slices can be warped via elastic deformation, to simulate data that is structurally different from the underlying dataset.

All augmentations are parameterized within certain bounds. Additional augmentations could theoretically be added to the pipeline with ease.

Sampling is primarily configured and executed by the `EMSampler` class.

2.4.2 Parallelization

For multi-core training environments, the framework parallelizes the sampling procedure, executing data sampling on multiple cores and adding the samples to a data queue, which can be sampled at each training step. This considerably speeds up training time when using GPUs, especially when the timing of the augmentation procedure is non negligible with respect to the timing of an optimization step.

2.5 Preprocessing

The framework enables the specification of preprocessing procedures to be executed before data flows into the Image Transformation stage of the pipeline. Currently the type of preprocessing procedures is limited to realignment using Spatial Transformers (anything that implements the `SpatialTransformer` interface) and standardization (a.k.a. whitening) of data, but it would be simple to implement additional preprocessing functionality.

2.6 Image Transformation

The framework allows clients to specify which type of image transformation should be included in the pipeline. Currently, the only types of image transformations that are directly implemented in the framework are variants of the Fully Convolutional Net and the U-Net[2]. In general, the Image Transform stage must take a 5-dimensional

[2]RNN-based Flood-Filling Networks may be available soon.

`Tensor` (the fundamental datastructure used in `TensorFlow`) with a shape of [batch-size, z-size, y-size, x-size, num-channels], and outputs a 5-dimensional `Tensor` with a shape of [batch-size, z-size, y-size, x-size, [1-3]] containing the predictions on the input data. The Image Transform must specify a `predict` function for inference.

2.6.1 Model Definition

There are only two broad classes of models currently supported: `ConvNet`(Fully Convolutional Nets), and `UNet`(U-Nets). However, the framework provides a set of primitives for each classes that allow for concise construction of nets with arbitrary structure, so long as they fit within the general paradigm of these two model types. These architectures are specified by both the `ConvArchitecture` and the `UNetArchitecture` classes, and are fed as parameters to the `ConvNet` and `UNet` classes for construction and automatic integration into the pipeline.

Particularly useful is that both classes of models automatically calculate the field-of-view of the models, and expose both the input and output shape to the pipeline so that at training time and inference time no extra specification or `Tensor`-wrangling must occur outside of the models. This means that to modify the architecture of a net, one need only change its respective `Architecture` specification, and nothing else. An example of an architecture specifcation for the 2-D N4 archetecture can be found below:

```
1 N4 = ConvArchitecture(
2     model_name='n4',
3     output_mode=BOUNDARIES,
```

```
    layers=[
        Conv2DLayer(filter_size=4, n_feature_maps=48, activation_fn=tf.nn.relu, is_valid=True),
        Pool2DLayer(filter_size=2),
        Conv2DLayer(filter_size=5, n_feature_maps=48, activation_fn=tf.nn.relu, is_valid=True),
        Pool2DLayer(filter_size=2),
        Conv2DLayer(filter_size=4, n_feature_maps=48, activation_fn=tf.nn.relu, is_valid=True),
        Pool2DLayer(filter_size=2),
        Conv2DLayer(filter_size=4, n_feature_maps=48, activation_fn=tf.nn.relu, is_valid=True),
        Pool2DLayer(filter_size=2),
        Conv2DLayer(filter_size=3, n_feature_maps=200, activation_fn=tf.nn.relu, is_valid=True),
        Conv2DLayer(filter_size=1, n_feature_maps=1, is_valid=True),
    ]
)
```

2.6.2 Model Training

Model training primarily occurs through the `Learner` class, which creates an optimizer based on a model's specified loss function (typically cross entropy), as well as various parameters specified in `TrainingParams`. Every step, the `Learner` feeds a training example from the queue into the model, runs the optimizer for one update step, and executes any number of user-specified `Hooks`. These hooks will execute

every N steps, where N is specified by the user in the hook constructor. Hooks provided include:

- `LossHook`: Report the loss for the model to `TensorBoard` every N steps.
- `ValidationHook`: Run inference on the validation set every N steps, and write both validation scores and image predictions to `TensorBoard`.
- `ModelSaverHook`: Save the model variables to disk every Nsteps, so that the model can be reloaded for inference.
- `HistogramHook`: Write distributions of the values of parameters for each `TensorFlow` variable to `TensorBoard` every N steps.
- `LayerVisualizationHook`: Write visualizations of the various feature maps for different convolutional layers to `TensorBoard` every N steps.

2.7 Postprocessing

Much like the preprocessing stage, the postprocessing stage of the pipeline allows for arbitrary transformations of the output of the Image Transformation stage. The only two transforms currently included in the framework are:

- Watershed: Given a set of specified parameters, convert a dataset annotated with affinities to a segmentation of the dataset. The current version is implemented in Julia.

- Mean Affinity Agglomeration: Given a segmentation and a set of affinities, greedily merge or split regions based on the affinity continuity along borders of the regions. The current version is also implemented in Julia.

2.8 Ensembling

The framework also enables the use of various ensembling techniques both at training time and at inference time through the `EnsembleLearner` class. This class allows a group of models to be trained simultaneously, and upon the completion of this training, an ensembling technique can be applied to their outputs for prediction. This ensembling technique can be any arbitrary ensembling method, including learned ensembling techniques that train on the outputs of various models. Currently the framework supports the following ensembling methods:

- `ModelAverager`: Average the output of several different models. If multiple copies of the same net are trained independently, averaging the outputs reduces the variance of predictions and leads to higher accuracy.

2.9 GPU Acceleration and Portability

Paramount in modern deep learning training is GPU Acceleration. In our experiments, using a GPU accelerated training speeds by factors of 100 or more, which was indispensible in the experimentation process. Because the entire pipeline sits on top of a `TensorFlow` backend, and `TensorFlow` automatically optimizes its own internal

processing graph for use on GPUs that support `CUDA`, our framework is GPU-enabled by default.

Because our group did not have a dedicated set of GPU hardware at the beginning of the project, we decided to use the containerization platform Docker, along with some NVIDIA plugins, to enable GPU training on any Linux machine with a GPU. By creating a Docker container that has the entire framework pre-installed, training and inference can be run on any machine that has access to a GPU with minimal setup.

Chapter 3

2D Segmentation

In this chapter, we establish the task of 2D Segmentation of EM images, attempt to train models that perform well on this task, and evaluate our results. The purpose of these experiments is not so much to achieve state-of-the-art performance on the task, but to examine the effect that increasing training data quality and reducing variance in predictions has on model performance.

3.1 Task Definition

2D Segmentation involves taking a single 2D slice of EM tissue and segmenting it into its constituent cells. Compared to 3D segmentation, this is a simple task, but will still allow us to show the properties of different models on EM data. To achieve segmentation, we will train models that predict boundaries of cells, and then use a Watershed algorithm to segment based on those boundary predictions.

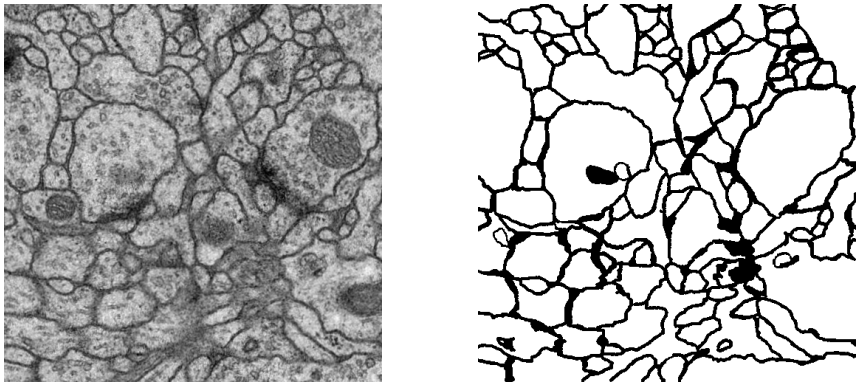

Figure 3.1: An example of 2D boundary detection. Left: the original image taken with an electron microscope. This particular example is neuron tissue taken from *Droposphila melanogaster* in a dataset created for the ISBI 2012 EM segmentation challenge [3]. The resolution of each pixel is 4nm x 4nm. Right: The ground truth boundaries corresponding to cell membranes in the input image, as labeled by human experts. The labels are binary values, although the actual border deliniation is somewhat arbitrary due to the fact that real applications of boundary detection are invariant to small differences in boundary shapes.

The problem statement for 2D Boundary detection is such: given a 2-dimensional single-channel (i.e. greyscale) image of neural tissue taken with an electron microscope, produce an image that labels the boundaries of all the distinct cells in the image. An example of this boundary-detection task can be found in Figure 3.1. This task is made somewhat more difficult by the existence of organelles with well-defined borders, as well as blood vessels and structured interstitial tissue.

3.2 Evaluation Metrics

The two main evaluation metrics we will use for this task are Rand Score and Pixel Error. Formal definitions of both of these error metrics can be found in Appendix A.

- **Rand Score**: We will use the Rand Score to determine whether or not the segmentation process correctly labels different cells as different objects. We will also look at the Rand Split Score and the Rand Merge Score, to see where the models inaccurately split and merge different regions.
- **Pixel Error**: We will use the Pixel Error to gauge the efficacy of our models at predicting the intermediate boundary stage.

3.3 Models

We define two models - both closely resembling models from the literature - with which we will run experiments:

- **N4**: The N4 Architecture is a standard, fully-convolutional network. We use an identical architecture to the architecture outlined in the original N4 paper[7]. The architecture uses comparatively fewer convolutions, each with comparatively larger filter sizes. The number of feature maps stays the same throughout the net, and ends with a fully connected layer to predict an output pixel with a sigmoid function. Intermediate non-linearities are ReLU. Given a large enough input, an entire output patch can be predicted pixel-by-pixel. The

effective field-of-view for each pixel in the output is a 95 pixel square in the input image.

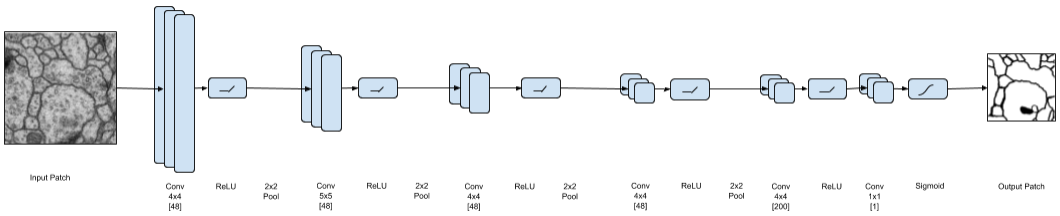

Figure 3.2: The N4 architecture used for 2D segmentation

- **VD2D**: The VD2D Architecture is also a standard, fully-convolutional network, and is nearly identical to the architecture originally used in the VD2D paper[15]. By comparison, the VD2D architecture is deeper than N4, and uses more successive layers with smaller convolutional filters. Activations are ReLU, and the output is a pixel affinity. The effective field of view for each pixel in the output is a 109 pixel square in the input image.

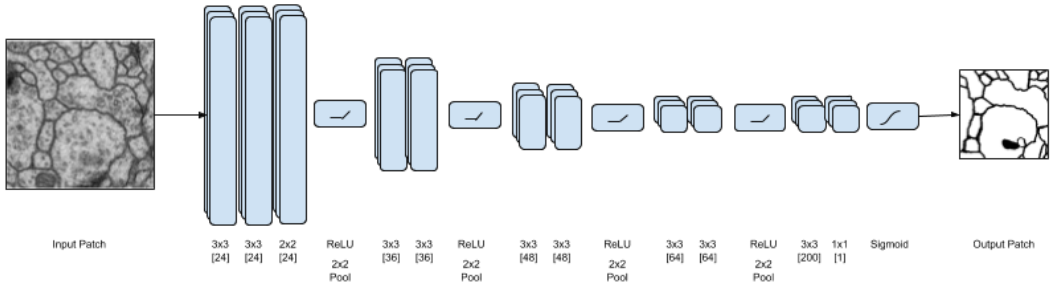

Figure 3.3: The VD2D architecture used for 2D segmentation

3.4 Dataset

One prominent competition that evaluates performance on this sort of task is the International Symposium on Biomedical Imaging (ISBI) EM Segmentation Challenge, which has had active submission since 2012. The ISBI Challenge organizers provides a training set of EM images, along with a set of binary boundary maps. The challenge website describes the training data as "a set of 30 sections from a serial section Transmission Electron Microscopy (ssTEM) data set of the Drosophila first instar larva ventral nerve cord (VNC). The microcube measures 2 x 2 x 1.5 microns approx., with a resolution of 4x4x50 nm/pixel"[3]. This resolution description implies that each pixel represents a 4x4nm patch on the surface of a slice, with each slice being 50nm thick. We build prediction systems using several different architectures, regularization methods, and data transformation techniques. We make several submissions to the leaderboard, ultimately scoring quite competitively.

The dataset comes with two EM stacks from the same neural tissue: one train stack with boundary labels consisting of (30) 512x512 slices; and one test stack without boundary labels consisting of (30) 512x512 slices. We hold out 25% of the train stack (7 slices) as a validation set to report results.

3.5 Training

We trained our nets each for 30000 iterations (with the exception of VD2D w/o augmentation, which was trained for 15000), sampling the training set in random patches, in mini-batches of 16. For every model, we sought to minimize the cross

entropy between the predicted boundaries and the true boundary labels, defining our loss function as:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) = -\log(\sigma(\boldsymbol{y}^T\boldsymbol{x} - (1 - \boldsymbol{y}^T)\boldsymbol{x}))$$

where σ is the sigmoid function, $\boldsymbol{x}$ is the prediction, and $\boldsymbol{y}$ is the true boundary.

At each step, we executed one iteration of optimization using the Adam Optimizer, and ever 100 steps we made predictions on the validation set to see how well the model generalized.

We made five different training runs using different parameters:

- **N4, w/o augmentation**: We trained our N4 model on the train data using the procedures above, making sure not to add any data augmentations during the training process.
- **VD2D, w/o augmentation**: We trained our VD2D model on the train data using the procedures above, making sure not to add any data augmentations during the training process.
- **N4**: We trained our N4 model on the train data using the procedures above, this time including data augmentations that randomly flipped, rotated, blurred, and otherwise distorted each sample before being fed into the net.
- **VD2D**: We trained our VD2D model on the train data using the procedures above, this time including data augmentations that randomly flipped, rotated, blurred, and otherwise distorted each sample before being fed into the net.

	Pixel Error	Rand - Full	Rand - Merge	Rand - Split
N4 w/o aug	0.112204	0.65693	0.506545	0.934315
N4	0.0827646	0.950296	0.933164	0.968069
VD2D w/o aug	0.0998995	0.80245	0.725266	0.898018
VD2D	0.0842352	0.954286	0.976404	0.933148
VD2D (x5)	0.083214	0.975745	0.985624	0.968843

Table 3.1: The results of various architectures on the 2D Segmentation task. Notice that using data augmentation drastically improves the performance of the nets. Additionally, ensembling multiple instances of the best architecture produces the best Rand Score.

- **VD2D (x5)** We independently ran 5 iterations of VD2D training (with augmentation), and then used the ensembling technique of model averaging to derive predictions.

3.6 Results

After training, we run the trained models on the validation set (since we don't have labels for the test set) to determine their performance. Results are numerically summarized in Table 3.1, and graphs of Rand Score and Pixel Error on the validation set over the course of training are displayed in Figure 3.4. Of the two models, VD2D generally performed better than N4, models trained with augmentation performed much better than those trained without, and the ensembled VD2D performed the best out of all training runs in terms of Rand Full Score.

Based on these results, we can draw three major conclusions. First, augmentation is extremely important in the training process. As the two graphs in Figure 3.4 demonstrate, models without random data augmentation learn for a short time,

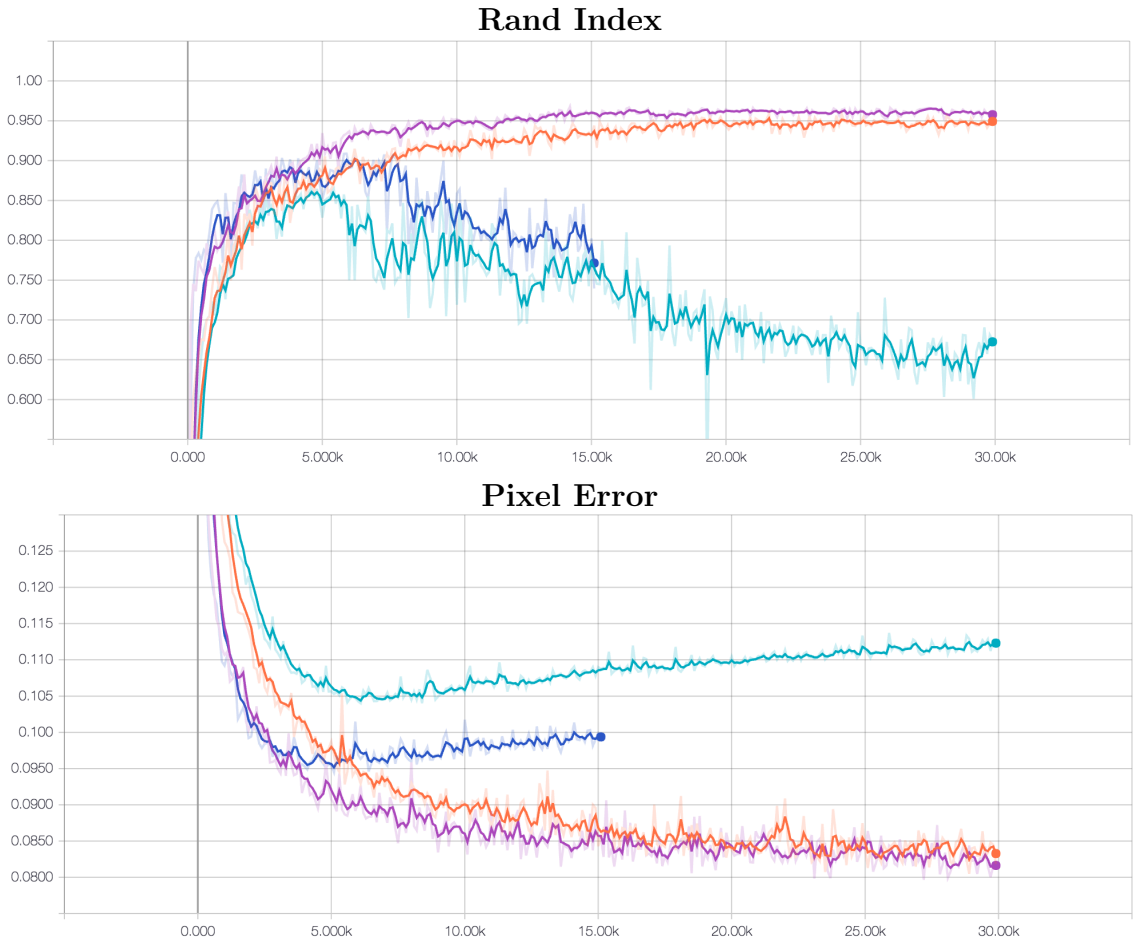

Figure 3.4: Training curves, smoothed, for 2D segmentation. Top: The full Rand scores on the validation set (from top: VD2D, N4, VD2D w/o augmentation, N4 w/o augmentation). Bottom: The pixel error on the validation set (from top: N4 w/o augmentation, VD2D w/o augmentation, N4, VD2D).

but they quickly begin to overfit, resulting in very poor generalization performance later on (even though the loss function continues to decrease). Second, we find that deeper nets can outperform shallower nets, since the deeper VD2D net outperforms the N4 architecture, which is a finding consistent with results of others who have used deep learning in segmentation tasks. Finally, we can conclude that these sorts of nets have a significant amount of variance in their predictions; by using ensembling methods we can reduce the variance between predictions, and thereby achieve higher performance. In later sections, we will explore further how training data quality affects training performance.

Chapter 4

3D Segmentation

In this chapter, we establish the task of 3D Segmentation of EM Images, attempt to train models that perform well on this task, and evaluate our results. The purpose of these experiments is not so much to achieve state-of-the-art performance on the task, but to examine the effect that increasing training data quality and reducing variance in predictions has on model performance.

4.1 Task Definition

The problem of 3D Segmentation is formulated as such: given a stack of 2-dimensional EM images generated that represent a 3-dimensional volume of tissue (i.e. the images were taken of successive physical slices of tissue), produce a segemen-tation[1] of the set of images that uniquely labels each discrete entity in the original

[1] A segmentation of an image or a stack of images is defined as producing a label for each pixel in the image or stack of images, where each unique label corresponds to a discrete object in the physical volume.

volume. That is, if a tissue volume contains a neuron that passes vertically through several different slices, then the portions of each slice through which the neuron passes would be labeled with the same identifier. This problem is significantly more complicated than the boundary prediction problem stated before, because it requires an awareness of context in 3 dimensions, rather than 2. Additionally, most EM datasets are anisotropic, meaning that the resolution is not uniform in all directions (specifically, the z-direction perpendicular to the plane of each image is generally dilated). An example of a segmentation can be found in Figure 4.1.

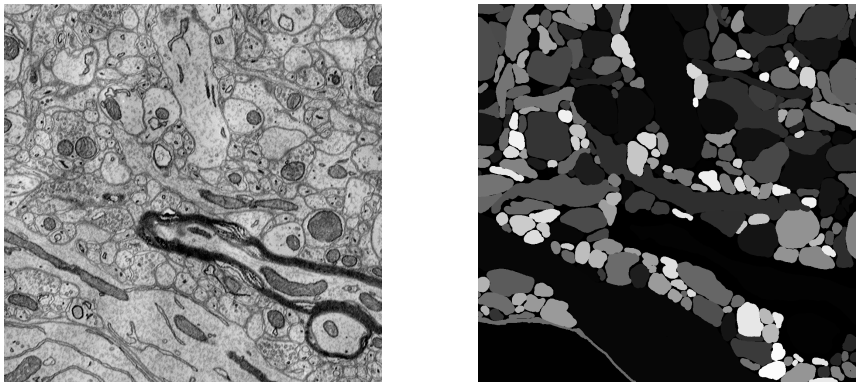

Figure 4.1: An example of a 2D cross-section of a 3D segmentation. Left: one of the original images in a stack of images taken with an electron microscope. This particular example is neuron tissue taken from the common mouse in a dataset used in the ISBI 2013 EM segmentation challenge [14]. The resolution of each pixel is 6nm x 6nm, and each image represents a slice 30nm thick. Right: The ground truth segmentation corresponding to a segmentation of each individual object in the input image, as labeled by human experts. The labels are unique identifiers, although the border deliniation is somewhat arbitrary due to the fact that real applications of boundary detection are invariant to small differences in boundary shapes.

Trivially, the complexity of objects in 3 dimensions is potentially much greater than in two dimensions, so it makes sense that any learning method used to train a system that performs segmentation might be adept at certain types of volumetric

data, and inept at others. To evaluate methods on different types of volumetric data, we selected two different challenges that provide us with samples of neural tissue that have different geometric properties, not only due to geometric differences in the underlying tissue but also because of differences in sample preparation techniques. These two challenges are the SNEMI3D Segmentation Challenge and the CREMI Segmentation Challenge.

4.2 Evaluation Metrics

Similar to the 2D Segmentation task, the two main evaluation metrics we will use for this task are Rand Score and Pixel Error. Formal definitions of both of these error metrics can be found in Appendix A.

- **Rand Score**: We will use the Rand Score to determine whether or not the segmentation process correctly labels different cells as different objects. We will also look at the Rand Split Score and the Rand Merge Score, to see where the models inaccurately split and merge different regions.

- **Pixel Error**: We will use the Pixel Error to gauge the efficacy of our models at predicting the intermediate boundary stage.

4.3 Models

We define two models for experimentation:

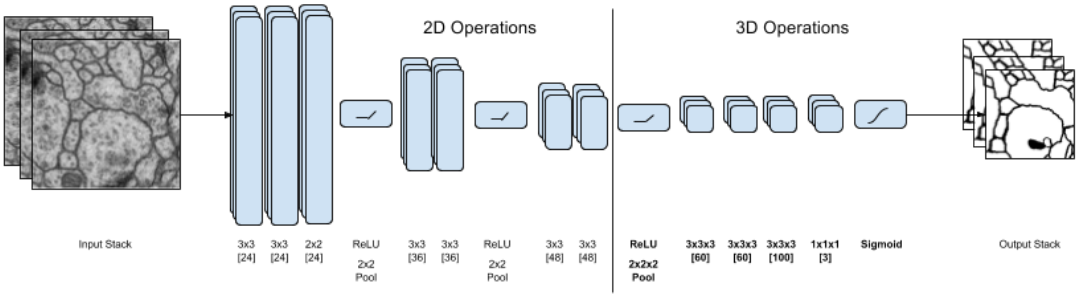

Figure 4.2: The VD2D-3D architecture for 3D Segmentation.

- **VD2D-3D**: The VD2D-3D implemented for this experiment is close to the one published by Lee et. al.[15]. It is a standard, fully-convolutional architecture that consists of two computational stages. The first stage of the net computes a series of 2D convolutions and poolings on each slice of a given sample stack, and the second stage computes a series of 3D convolutions and poolings on the entire stack. Intermediate nonlinearities are ReLU. Because the dataset is anisotropic, the effective field of view in the z direction is much smaller than in the x and y direction in terms of pixels, which is achieved through fewer z-dimensional poolings and smaller z-dimensional convolutions. The model predicts a set of 3 affinities (x, y, z) for each pixel. The effective field of view is a 85 pixel square in the x-y plane, and 7 pixels in the z direction.

- **UVR-Net**: The UVR-Net borrows concepts from the U-Net, V-Net, and Residual Nets, and is an architecture that all members of our group contributed to[22, 10, 19, 6]. The structure consists of 4 "U Layers", which are pairs of down-convolutions and up-convolutions. The output of the down-convolutions skips across the net, and is concatenated to the input of the corresponding up-

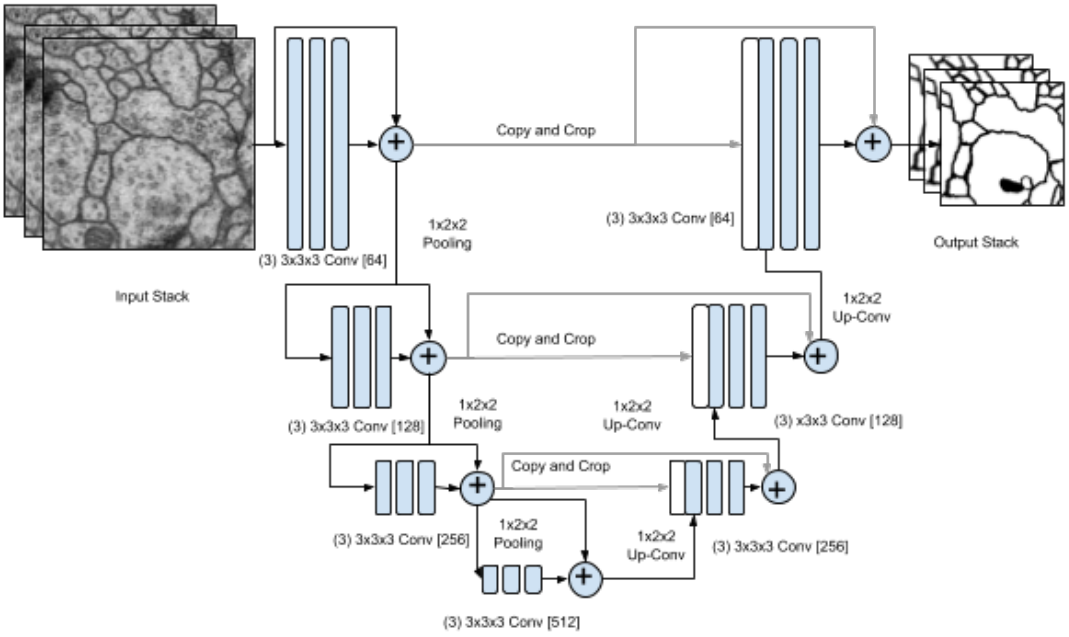

Figure 4.3: The UVR-Net architecture for 3D Segmentation

convolutional component (the number of feature maps of the first convolution in each "U Layer" is twice the subsequent ones). We add residual connections that skip the convolutions from the output of pooling layers, and add skip connections across the "U-Layer". After each convolution, we apply ReLU nonlinearities, except between convolutions and additions with residual connections. These modifications are all added to attempt to make the net learn first by predicting an output that looks similar to the input, and slowly eroding elements until an affinity map is left.

4.4 Dataset

We will train and experiment on (4) 3D EM Datasets, taken from two different competitons: SNEMI3D and CREMI. We evaluate on 4 datasets instead of 1 to discuss the shortcomings of these models, and how training on different types of data affect performance.

The SNEMI3D Segmentation Challenge is a highly active 3D segmentation challenge (organized in advance of ISBI 2013), and provides a stack of EM images for training, along with ground truth segmentations of the EM images in 3 dimensions. The challenge website describes the training and testing data as "stacks of 100 sections from a serial section Scanning Electron Microscopy (ssSEM) data set of mouse cortex. The microcube measures 6 x 6 x 3 microns approx., with a resolution of 6x6x30 nm/pixel"[2]. Like the ISBI 2012 dataset, the SNEMI3D dataset is anisotropic, and particularly dilated in the z-direction. Additionally, the data is from mouse cortex, rather than from *Droposphilia*, and the geometry of the tissue is significantly different.

The SNEMI3D dataset comes with a train and a test set, both of which consist of (100) 1024x1024 pixel images. The train set includes a set of labels, which represent a segmentation of the bodies in the stack. Since the test set does not include such a segmentation, we create a validation set of 25 images (25%) which we will use to evaluate the performance of our models.

The Circuit Reconstruction from Electron Microscopy Images (CREMI) Challenge is a somewhat less-active challenge organized in advance of MICCAI 2016[8]. The challenge provides three datasets for training, all of which are volumetric samples

of *Drosophila melanogaster*. The training and testing data are stacks of 125 sections from an ssSEM data set, with each slice having a resolution of 4x4x40nm/pixel. These datases are also anisotropic, being dilated in the z-direction. Furthermore, the types of neurons sampled are quite diverse between datasets: from visual inspection, some of the neurites in one of the datasets is much thinner than those in the others, suggesting that models might perform differently when trained/tested on these different datasets. Finally, these datasets are quite a bit noisier than ISBI or SNEMI3D: there are many more major misalignments, many patches of blur, and some slices are missing entirely. These datasets will provide a good measure of how robust our methods are to noise in volumetric data.

The three CREMI datasets (labeled CREMI A, CREMI B, and CREMI C) contain train and test sets, each of which consist of (125) 1250x1250 pixel images. The train set includes a set of labels, which represent a segmentation of the bodies in the stack. Like SNEMI3D, the test sets don't include segmentaitons, so we create validation sets from the training sets by witholding 25 slices (20%) from each for evaluation of model performance.

4.5 Training

We train each model on each of the datasets, for a total of 8 training runs. Training was performed on a NVIDIA Titan X. We trained each net for 30000 iterations, sampling volumetric sections from each dataset that were 16 slices thick. For every

model, we sought to minimize the cross entropy between the predicted affinities and the true afinnity labels, defining our loss function as:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) = -\log(\sigma(\boldsymbol{y}^T\boldsymbol{x} - (1 - \boldsymbol{y}^T)\boldsymbol{x}))$$

where σ is the sigmoid function, $\boldsymbol{x}$ is the prediction, and $\boldsymbol{y}$ are the true affinites.

At each step, we executed one iteration of optimization using the Adam Optimizer, and ever 100 steps we made predictions on the validation set to see how well the model generalized.

4.6 Results

After training, we run prediction on the validation sets for each dataset (since we don't have labels for the test set) to determine their performance. Results are numerically summarized in Table 4.1. The training graphs documenting Rand Score and Pixel Error for models training on SNEMI3D, CREMI A, CREMI B, and CREMI C can be found in Figures 4.4, 4.5, 4.6, and 4.7, respectively.

We can draw several conclusions about the qualities that these models have when trained on these datasets. First, it is patently obvious that there is large variation in the training process. Sometimes, the models would train quite quickly; sometimes, they wouldn't train appropriately (in the case of VD2D-3D on CREMI C). If we were to repeat this study, we would likely train each model on each dataset multiple times with the same parameters, and take the best performing model from each set of runs.

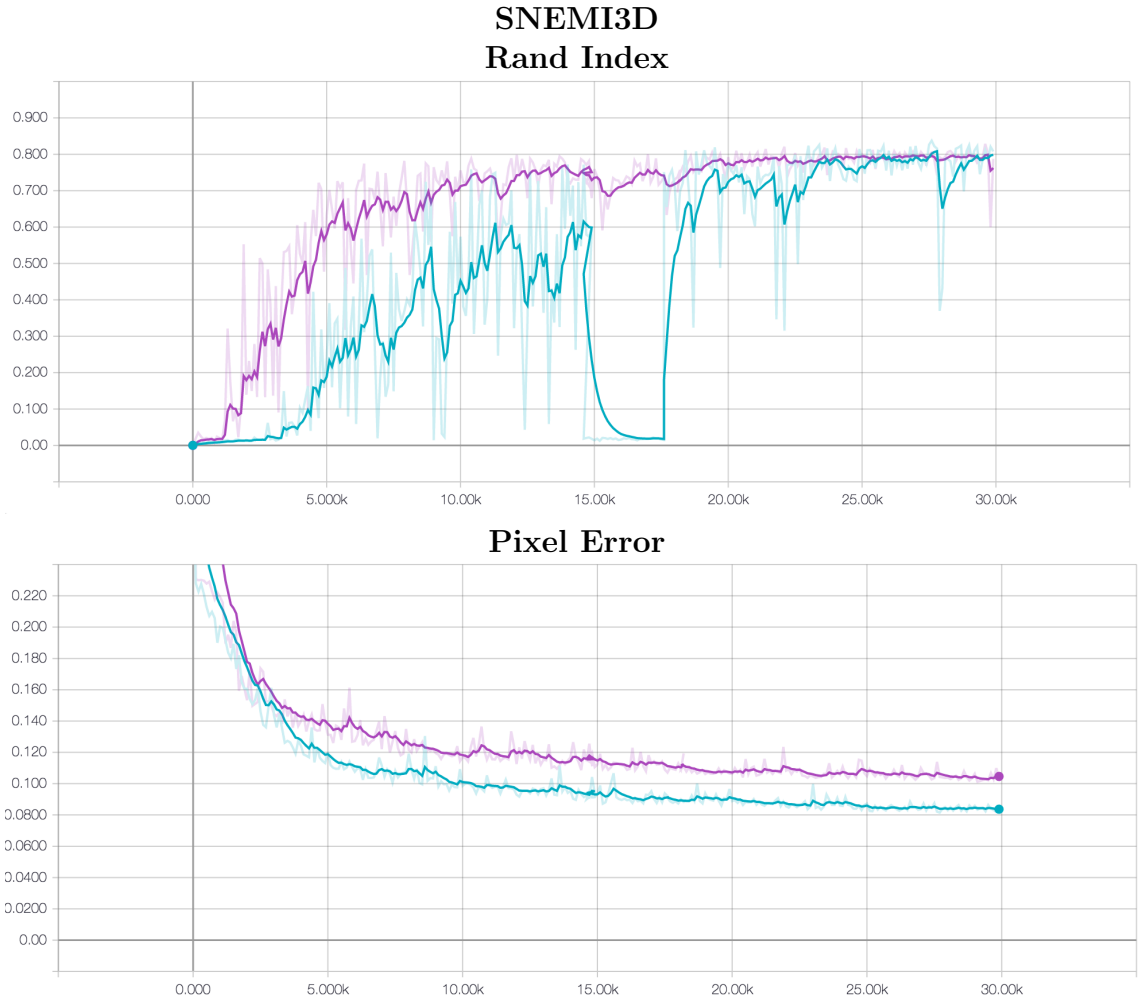

Figure 4.4: Training curves, smoothed, for 3D segmentation. Top: The full Rand scores on the SNEMI3D validation set (from top: VD2D-3D, UVR-Net). Bottom: The pixel error on the SNEMI3D validation set (from top: VD2D-3D, UVR-Net). The sharp dip in the Rand Score for the UVR-Net was due to a small parameter change in validation procedure that was quickly reverted.

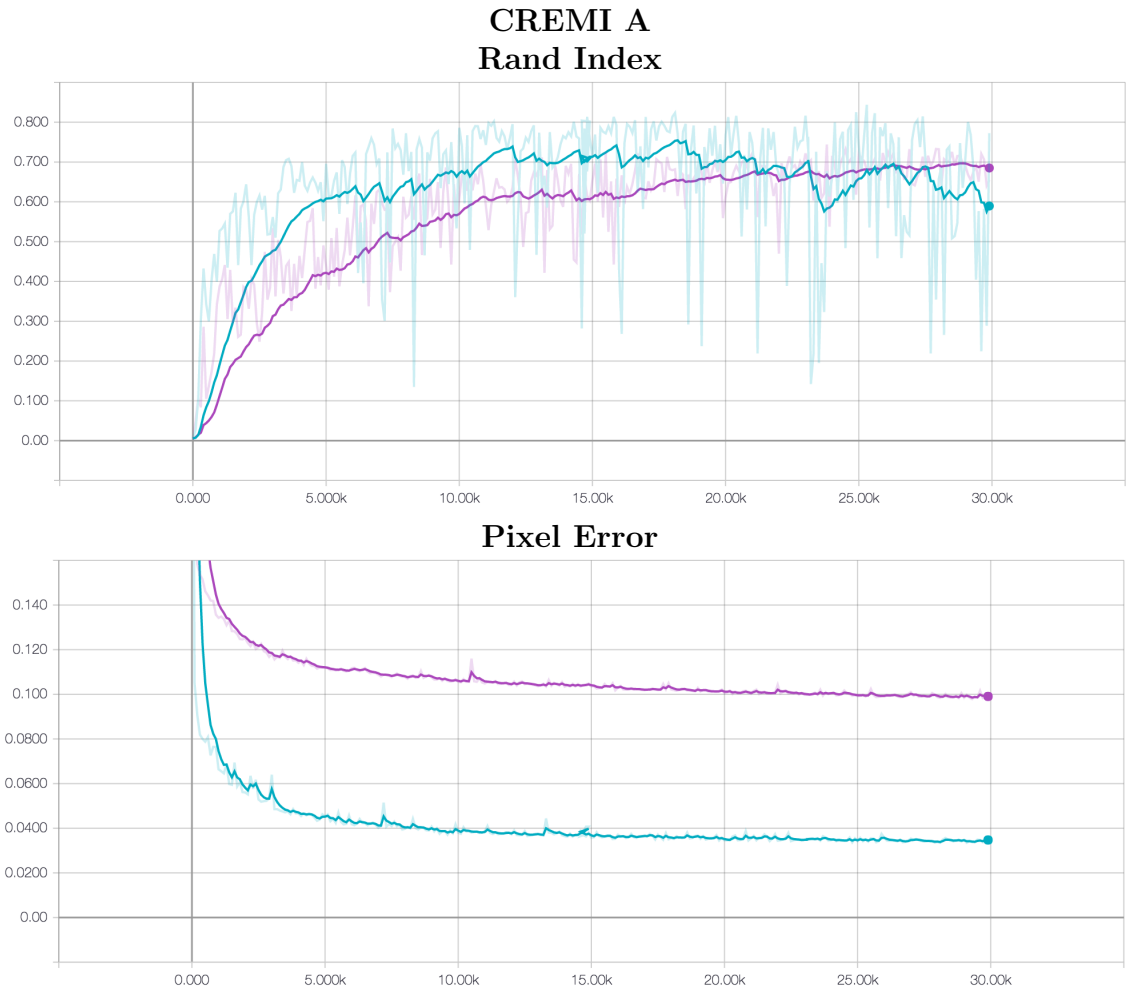

Figure 4.5: Training curves, smoothed, for 3D segmentation. Top: The full Rand scores on the CREMI A validation set (from top: UVR-Net, VD2D-3D). Bottom: The pixel error on the CREMI A validation set (from top: VD2D-3D, UVR-Net).

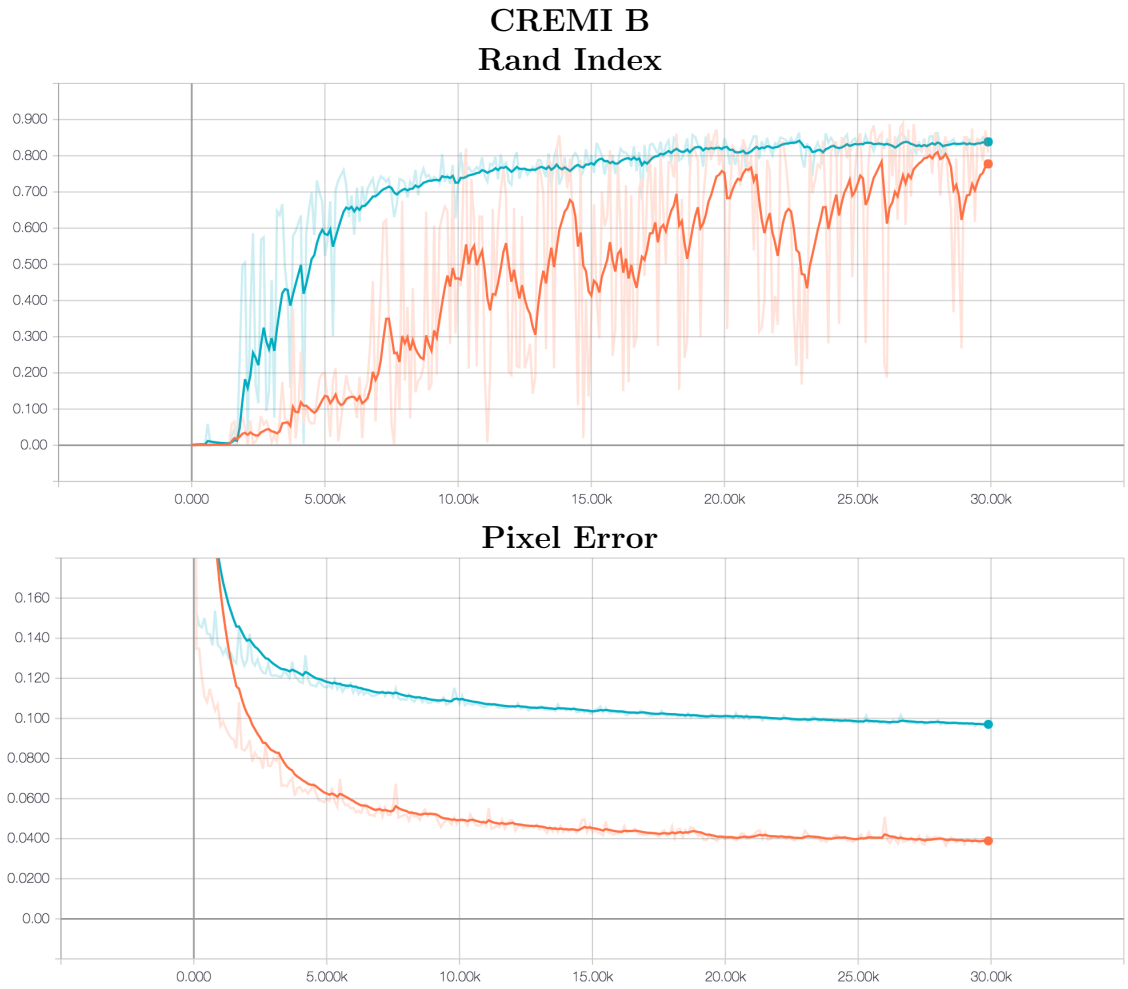

Figure 4.6: Training curves, smoothed, for 3D segmentation. Top: The full Rand scores on the CREMI B validation set (from top: VD2D-3D, UVR-Net). Bottom: The pixel error on the CREMI B validation set (from top: VD2D-3D, UVR-Net).

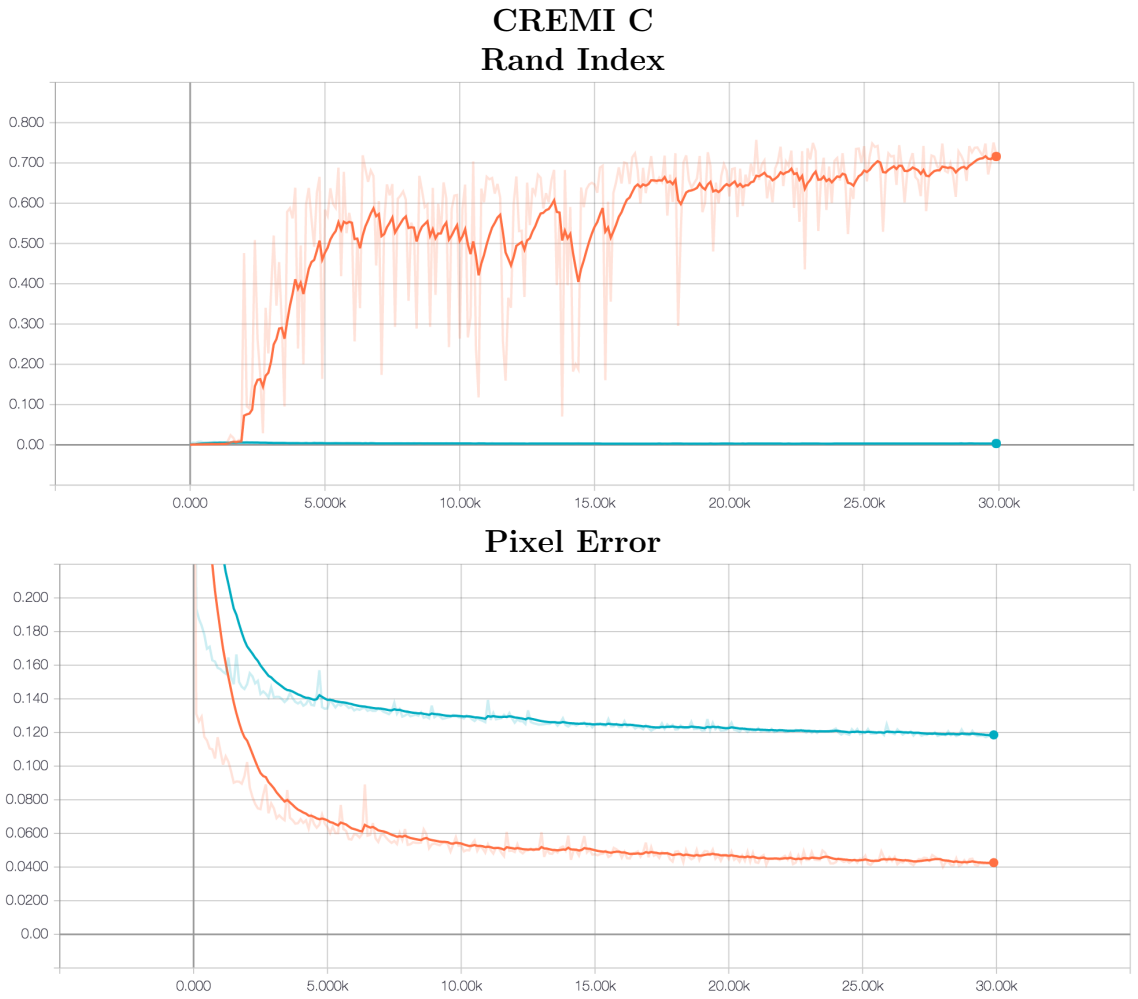

Figure 4.7: Training curves, smoothed, for 3D segmentation. Top: The full Rand scores on the CREMI C validation set (from top: UVR-Net, VD2D-3D (barely pictured)). Bottom: The pixel error on the CREMI C validation set (from top: VD2D-3D, UVR-Net). The absurdly low Rand score for the VD2D-3D model might be a bit misleading; it is possible that there was a training fluke for this iteration.

Table 4.1: Results of 3D Segmentation on various datasets

SNEMI3D

	Pixel Error	Rand - Full	Rand - Merge	Rand - Split
VD2D-3D	0.102204	0.790909	0.95196	0.676465
UVR-Net	0.0863271	0.783288	0.993469	0.646511

CREMI A

	Pixel Error	Rand - Full	Rand - Merge	Rand - Split
VD2D-3D	0.098559	0.651105	0.981541	0.487117
UVR-Net	0.034879	0.576646	0.449545	0.803948

CREMI B

	Pixel Error	Rand - Full	Rand - Merge	Rand - Split
VD2D-3D	0.097363	0.845614	0.929446	0.775653
UVR-Net	0.0379261	0.846339	0.933393	0.774138

CREMI C

	Pixel Error	Rand - Full	Rand - Merge	Rand - Split
VD2D-3D	0.11817	0.00283098	0.91909	0.00141767
UVR-Net	0.0419219	0.748079	0.933252	0.624223

Second, in every setting the UVR-Net outperforms the VD2D-3D net in Pixel Error; UVR-Net consistently has substantially lower pixel error than VD2D-3D, by about the same margin in every trial. This suggests that the residual layers and skip connections built into the architecture are doing their jobs, and are reconstructing images that look a lot like the input (an affinity map should look quite like the boundaries of the input).

Thrid, the quality of the data significantly affects the performance of these models. While some of the differences in scores can be attributed to variance in the

training of the model, during the experimentation process (before results generation) it became clear that models would perform differently on different datasets. From this, we can draw the conclusion that certain datasets are "harder" than others to learn segmentatons from. What makes one dataset "harder" than another?

In the SNEMI3D dataset, for instance, many of the objects are significantly thinner and extend longer distances across the EM slices than in the other datasets. Data quality can make a huge difference as well. The three different CREMI datasets, although taken from the same biological sample and imaged with the same process, have different properties in terms of stack quality. Visually looking at each layer of the CREMI stacks, the quality of the images varies greatly. Some of the CREMI datasets have significantly more blurred slices than others; the same goes for blemishes and cracks. CREMI C is missing a slice entirely. Individual slice artifacts may have a small impact on final metrics, but it is likely that only dataset errors affecting many slices across a dataset will have significant impact on the Rand Score of predictions. The most obvious culprit between the CREMI datasets is alignment. All three datasets are poorly aligned across the entire stack, but in terms of alignment CREMI A is worse than CREMI B and CREMI C. The Rand scores on CREMI A were also lower across all models than the other CREMI datasets. While this is not concrete evidence that the models are quite susceptible to error on poorly aligned datasets, the correlation certainly suggests that improving alignment procedures for EM datasets, or even building explicit alignment layers into models might produce better results across diverse datasets.

Chapter 5

Alignment

One major hurdle in inferring neural structure from EM images is that the image acquisition process is inherently noisy. While the EM imaging technologies used for the creation of neuron images (typically TEM) are quite stable, there is often variance in sample preparation techniques, resulting in all sorts of distortions and errors at imaging time. One particular type of error, image misalignment, occurs during the slicing of sample tissue, when some physical factor causes a resulting slice to be warped or translated in such a way that the resulting stack of images is misaligned. Intuitively, this means that every point in one EM slice data does not necessarily map to the point directly below it the neighboring slice. An example of slice misalignment can be visualized in Figure 5.1.

The problem of misalignment within a set of EM images particularly induces problems in the task of 3D Segmentation. While most techniques are rather invariant to small misalignments (particularly CNNs, which can be trained to be invariant

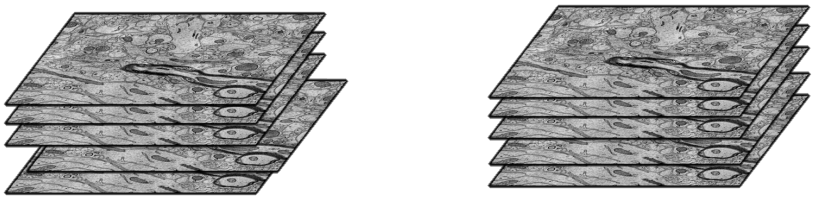

Figure 5.1: An example of a 3D stack of EM images that contains a misalignment. Left: The provided alignment of a stack. This represents a misalignment where the fourth image in a stack of images actually represents a slice slightly translated in position. Right: The correct alignment of the stack, where all the pixels in the fourth position have been translated enough such that the structures depicted in the input data line up in the z-direction.

to warping of many kinds), large misaligments can often induce false splitting in segmentations. Very deep CNNs trained with a really diverse set of data would likely be able to compensate for these sorts of misalignments, but it would be more prudent to develop a more efficient strategy for automatically healing misalignments in the data.

In this section, we will define the task of realignment, construct and train models, and examine the results of these experiments.

5.1 Task Definition

In an abstract sense, the realignment task is to take a stack of raw EM images and distort it such that it as accurately as possible reflects the reality of the underlying biological structures it supposedly represents. This is a bit too daunting of a task for

the scope of this paper, so instead propose the task of realigning one 'distorted' slice to another 'reference' slice, with the stipulation that these two slices are adjacent within some EM stack. That is, transform one of two consecutive images in an EM stack such that their aligment is maximized.

5.2 Evaluation Metrics

Coming up with an empirical metric to measure the aligment of two images is difficult, because domain-knowledge is required to say with certainty whether two images are probably aligned. So, in lieu of a metric for alignment, we will take an already aligned (i.e. by the empirical methods mentioned in Chapter 1) stack of images and randomly distort them. Both the parameters for distortion (which can be any sort of parameterized transform) and the original, undisturbed image of the 'distorted' slice can serve as labels when evaluating error, since the distortions were induced.

From this formulation, we define three evaluation metrics, all of which are defined in Appendix A:

- **Pixel Error**: We will use Pixel Error to determine whether, when given a correctly-aligned 'reference' image and a 'distorted' image, the model outputs an image that is pixelwise close to the preimage of the 'distorted' image.
- **Cross Correlation**: Similar to Pixel Error, we will use Cross Correlation to determine whether or not a model's prediction aligns with the preimage of a 'distorted' image. Cross Correlation is somewhat more continuous than Pixel Error.

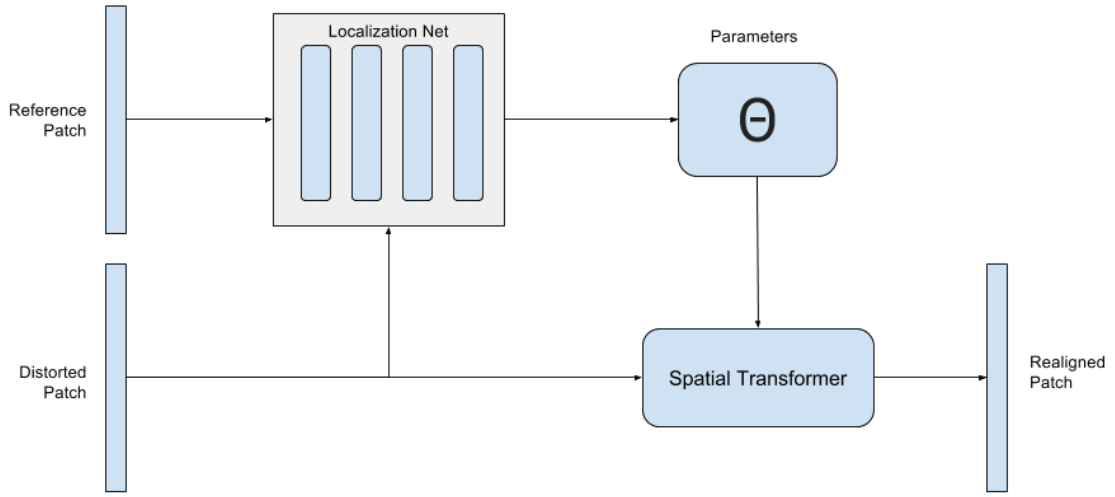

Figure 5.2: A prototypical Spatial Transformer Network. The major components of the network are the localization net, and the spatial transformer module. Both sections are fully differentiable, meaning that a gradient can be backpropogated through the transform and up through the localization net for training.

- **L2 Error on Parameters**: The parameters to a transformation are typically continuous real numbers, meaning that we can calculate a meaningful L2 Error on the Parameters if our model predicts a set of parameters (rather than simply predicting the set of images).

5.3 Models

Both of our models for predicting realignment are based on the Spatial Transformer Network, which was originally designed as a tool to regularize convolutional networks and speed up image classification tasks[11]. Spatial Transformer Networks are conceptually straightforward: it is possible to define a class of image transformations

(i.e. rotation, translation, affine) that are at least partially differentiable. If a transformation is at least partially differentiable, then any gradient that is calculated at the output of the transformer can be backpropogated through the transformer and into its parameters. This means that parameters can be learned, through the use of a localization net. In general, for a chosen spatial transform (a sub-differentiable transform), localization nets can take on any architecture that produce appropriate parameters for that transformer. We define two models; for both we will choose as our spatial transformer module the affine transform[1] module as outlined in the original paper. Our two localization models are:

- **Fully Connected**: The Fully Connected (FC) localization network is a shallow attempt at learning simple transformations. For a pair of fixed size images with N pixels (the 'reference' image and the 'distorted' image), the FC localization network has two fully connected layers, with 512 units in the first layer, and 6 units in the last layer. We apply ReLU between the first and second layer, and because the parameters of an affine transform can be negative or positive, the final layer is run through a tanh function before being output.

- **Standard Convolutional**: The Standard Convolutional localization network has the same set of outputs and inputs as the FC localization network, namely taking two N-sized inputs, and outputting 6 parameters. The network consists of 4 convolutional layers, each with 3 (1x3x3) convolutions into 48 feature maps, a ReLU nonlinearity, and a (1x2x2) pooling. After these four layers are two

[1] An affine transform is a class of transforms that include rotation, scaling, shearing, and translating. An affine transform is parameterized by 6 real numbers.

fully connected layers, that work in the same way as in the FC localization net, except with inputs that match the outputs of the last pooling.

Both these models were chosen because of examples in the original Spatial Transformer paper. Future work will likely consist of designing alternate architectures.

5.4 Dataset

For simplicity, we use the ISBI 2012 dataset referenced in Chapter 3. This dataset is quite well-aligned, and serves as a good ground-truth for random misalignments to be applied to. Just as before, a validation set was held out from the training set to give us a way to evaluate our performance.

5.5 Training

We trained our models for 100,000 steps each, randomly sampling a pair of adjacent patches from the dataset and applying a random translation or rotation to both patches, designating one patch as the 'reference' patch and the other the 'distorted' patch. A third 'ground truth' patch is calcuated by taking the preimage of the 'distorted' patch and applying the same transformation as was applied to the 'reference' patch. This creates a training input and label. Both translation and rotation were bounded to within realistic values for the problem domain (i.e. so that images weren't offset by twice their length).

We attempted to train each of our models under various conditions: without augmentation on the dataset, and with augmentation on the dataset when sampled. That is, we either drew from a very small training set, or a very large one.

We attempted to use two different loss functions: smoothed cross correlation, and L2 Loss on the parameters. The smoothed cross correlation was calculated after the predicted transformation was applied, but because both of our model types predicted parameters for a transformation using a Spatial Transformer, we were able to backpropogate the gradient for this loss function back through the Spatial Transformer and into the predicted parameters.

5.6 Results

Unfortunately, we found that our quantitative results were not of sufficient quality to report. We found that when training both styles of localization nets without augmentation the nets would actually learn how to compensate for arbitrary translations and rotations on data it had seen before. That is, if it saw a pair of patches it had seen before, but not necessarily with the particular transformation, it could heal that never-before-seen transformation. However, there was no generalization whatsoever when testing on the validation set: predictions were effectively random. This occured for all of our models, regardless of whether we trained with the pre-transformation parameters as our label or with the transformed image as our label.

Despite these discouraging results, we believe that this particular avenue for exploration has promise. Because we were able to train nets that could learn arbitrary

transformations on image data it had seen before, we believe that the primary impediment is localization net architecture, rather than the whole Spatial Transformer concept. Perhaps a deeper or wider net would show improvements, or one with residual connections to maintain the original features of the 'distorted' image.

Additionally, one of the key properties of the Spatial Transformer Network is that it can propogate gradients from downstream tasks through the transformation layer. This could potentially allow for a segmentation model that, instead of learning based ona fixed alignment, dynamically learns an alignment such that segmentation accuracy is directly maximized.

Chapter 6

Conclusion

If there is one thing that we have learned throughout the experimentation process, it's that making incremental improvements to any stage of the EM Segmentation pipeline is a both a large theoretical and computational challenge. Because there is not a strong theoretical underpinning for the emergent properties of various neural networks, it is often quite difficult to come up with an architecture that learns anything about segmentation, let alone outperforms some existing state-of-the-art model. At the same time, the data processing pipeline becomes more complex as more advanced methods are implemented, and it is inevitable that experimentation velocity will diminish with the number of parameters to tune in the system. It is this duality of problem that inspired this thesis, and now brings us to our conclusion.

If the topics covered in Chapters 1-5 appeared to conceptually follow a logical progression, that was intended. In Chapters 1 and 2, we discussed the practical problems of developing a real-world system for EM Segmentation, and explored various

architectural and design solutions to these problems. Our own struggle as researchers with these problems culminated in the development of the presented EM Segmentation framework, `DeepSeg`. While the `DeepSeg` framework is far from perfect, it allowed us to build meaningful abstractions as we explored different architectures and pipeline components without worrying too much about how different parts of the pipeline were affected by our explorations. Moreover, it allowed us to easily repeat experiments and keep track of the results we generated.[1]

Armed with this framework, in Chapter 3 we were able to demonstrate some of the various parameters and dataset attributes that affect the performance and generalizability of models on 2D segmentation. While the problem of 2D segmentation is not particularly exciting, as state-of-the-art models perform on par with human experts, it is a good problem domain in which to concretely explore the effects caused by altering these parameters and dataset attributes. To this end, we showed that deeper networks performed better than shallower ones, and that substantial data augmentation is extremely important to the generalizability of models.

We expanded on the idea of dataset quality in Chapter 4, where we explored the challenge of 3D segmentation on 4 separate datasets. While the original intent of the research project was to build self-contained models that would approach or exceed state-of-the-art methods for 3D EM segmentation, we quickly discovered that not all 3D EM segmentation tasks are not equal. Some datasets are better than others. We did show that there was variance in both the models we trained, and that adding skip connections and residual layers helped enforce some invariants in the look of the

[1] In fact, all of the results generated in this paper were generated by repeatable experiments set up in `Jupyter` notebooks alongside the text of this report.

output. But the major takeaway from this section is that imaging artifacts, and in particular misalignments, have a major impact on the performance of several classes of models, even if the underlying structures of the data are roughly the same.

In Chapter 5, we attempted - albeit unsuccessfully - to explore automated methods for correcting misalignments in EM stacks using learned methods. We were particularly concerned with Spatial Transformer Networks, which rather than learn an image transformation from scratch simply learn the parameters to a pre-defined image transform, doing so in such a way that error can be propogated up from a downstream task to inform better transforms. While our attempts to apply these techniques to EM stack realignment were not emperically successful, certain results pointed to the notion that while our attempts were incorrect, the approach in general might yield positive results with more experimentation.

Now, at the conclusion of this thesis, we are now concerned with the implications of our findings. While we did not develop models that achieved state-of-the-art performance in any of the benchmark tasks - we preform admirably, but not at the top of the leaderboards - we developed a set of tools and model paradigms that are good starting points for future research in various areas of the EM segmentation pipeline. In terms of research direction and future work, we are most interested in developing segmentation models that are end-to-end trainable. That is, we are interested in developing an EM segmentation pipeline where each stage is fully (or at least partially) differentiable, allowing direct training of segmentations from raw inputs. While there is still a monumental amount of research requred to create such a pipeline that performs effectively, we believe that replacing the various hand-tuned

components of the pipeline with trained components is a promising avenue of research for boosting end-segmentation performance.

Appendix A

Metric Definitions

A.1 Pixel Error

Given two images, pixel error is defined as the mean of the absolute numerical distance between corresponding pixels in both images. Formally, we define:

$$P(X, Y) = \frac{\sum_{\boldsymbol{i} \in S} |X_{\boldsymbol{i}} - Y_{\boldsymbol{i}}|}{|S|}$$

for tensors X and Y over all indexes $\boldsymbol{i}$ in the index space S for tensors X and Y.

A.2 Rand Score

The version of the Rand Score used in this thesis is the Rand F Score. The Rand F Score essentially counts all the non-distinct pairs in an image that are correctly labeled as belonging to the same or different grouping with respect to a reference

image. Formally, we define $S_1, S_2, ...S_n \subseteq S$ to be the set of all groupings S_i in an input volume (i.e. distinct labels in a segmentation)m and $T_1, T_2, ...T_n \subseteq T$ to be the set of all groupings T_i in a ground-truth volume. Let:

$$t_i = |T_i|$$

$$s_i = |S_i|$$

$$c_{i,j} = |S_i \cap T_j|$$

Then, the Rand F Score can be defined as:

$$R_{\mathsf{Full}}(S,T) = \frac{\sum_{i,j} c_{i,j}}{\alpha \sum_i s_i^2 + (1-\alpha) \sum_j t_i^2}$$

$$R_{\mathsf{Merge}}(S,T) = \frac{\sum_{i,j} c_{i,j}}{\sum_i s_i^2}$$

$$R_{\mathsf{Split}}(S,T) = \frac{\sum_{i,j} c_{i,j}}{\sum_j t_j^2}$$

where N is the number of voxels in a volume. See `https://github.com/seung-lab/segascorus/blob/master/segerror-manual.pdf` for more details of the Rand F Score.

A.3 Cross Correlation

Given two images, cross-correlation is defined as the mean of the prodiuct of corresponding pixels in both images. Formally, we define:

$$C(X, Y) = \frac{\sum_{\boldsymbol{i} \in S} |X_{\boldsymbol{i}} \times Y_{\boldsymbol{i}}|}{|S|}$$

for tensors X and Y over all indexes $\boldsymbol{i}$ in the index space S for tensors X and Y.

A.3.1 Smoothed Version

Because standard Cross Correlation is not particularly continuous (i.e. two images will typically be highly correlated if they match exactly, and loosely correlated if they are translated or rotated), we define a somewhat more smoothed version that attempts to make the function more continuous. We define the function:

$$C_s(X, Y) = \frac{\sum_{\boldsymbol{i} \in S} |X_{\boldsymbol{i}} \times (G_\theta(Y))_{\boldsymbol{i}}|}{|S|}$$

for tensors X and Y over all indexes $\boldsymbol{i}$ in the index space S for tensors X and Y. The function $G_\theta(\cdot)$ is a smoothing function, which applies a Gaussian filter to its arguments based on parameters θ. The amount of smoothing that occurs determines how dontinuous the smoothed cross-correlation function is. Because the continuity of this function can be controlled, it makes for a more-useful loss function than standard Cross Correlation.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, Xiaoqiang Zheng, and Google Research. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.

[2] Ignacio Arganda-Carreras, H. Sebastian Seung, Ashwin Vishwanathan, and Daniel R. Berger Vishwanathan. ISBI 2013 challenge: 3D segmentation of neurites in EM images — SNEMI3D: 3D Segmentation of neurites in EM images, 2013.

[3] Ignacio Arganda-Carreras, Srinivas C. Turaga, Daniel R. Berger, Dan Cirean, Alessandro Giusti, Luca M. Gambardella, Jürgen Schmidhuber, Dmitry Laptev, Sarvesh Dwivedi, Joachim M. Buhmann, Ting Liu, Mojtaba Seyedhosseini, Tolga Tasdizen, Lee Kamentsky, Radim Burget, Vaclav Uher, Xiao Tan, Changming Sun, Tuan D. Pham, Erhan Bas, Mustafa G. Uzunbas, Albert Cardona, Johannes Schindelin, and H. Sebastian Seung. Crowdsourcing the creation of image segmentation algorithms for connectomics. *Frontiers in Neuroanatomy*, 9:142, nov 2015.

[4] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *Cvpr 2015*, page 5, 2015.

[5] Thorsten Beier, Constantin Pape, Nasim Rahaman, Timo Prange, Stuart Berg, Davi D Bock, Albert Cardona, Graham W Knott, Stephen M Plaza, Louis K

Scheffer, Ullrich Koethe, Anna Kreshuk, and Fred A Hamprecht. Multicut brings automated neurite segmentation closer to human performance. *Nature Methods*, 14(2):101–102, jan 2017.

[6] Özgün Çiçek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation. jun 2016.

[7] Dan C Cirean, Alessandro Giusti, and Luca M Gambardella. Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images.

[8] Funke. Jan, Stephan Saalfeld, Davi Bock, Srini Turaga, and Eric Perlman. CREMI: Circuit Reconstruction from Electron Microscopy Images, 2016.

[9] Robert M. Haralick and Linda G. Shapiro. Image segmentation techniques. *Computer Vision, Graphics, and Image Processing*, 29(1):100–132, jan 1985.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *Arxiv.Org*, 7(3):171–180, 2015.

[11] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and koray Kavukcuoglu. Spatial Transformer Networks, 2015.

[12] Michał Januszewski, Jeremy Maitin-Shepard, Peter Li, Jörgen Kornfeld, Winfried Denk, and Viren Jain. Flood-Filling Networks. nov 2016.

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

[14] Narayanan Kasthuri, Kenneth Jeffrey Hayworth, Daniel Raimund Berger, Richard Lee Schalek, Jos?? Angel Conchello, Seymour Knowles-Barley, Dongil Lee, Amelio V??zquez-Reina, Verena Kaynig, Thouis Raymond Jones, Mike Roberts, Josh Lyskowski Morgan, Juan Carlos Tapia, H. Sebastian Seung, William Gray Roncal, Joshua Tzvi Vogelstein, Randal Burns, Daniel Lewis Sussman, Carey Eldin Priebe, Hanspeter Pfister, and Jeff William Lichtman. Saturated Reconstruction of a Volume of Neocortex. *Cell*, 162(3):648–661, 2015.

[15] Kisuk Lee, Aleksandar Zlateski, Ashwin Vishwanathan, and H Sebastian Seung. Recursive Training of 2D-3D Convolutional Networks for Neuronal Boundary Detection.

[16] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. nov 2014.

[17] D.G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, pages 1150–1157 vol.2. IEEE, 1999.

[18] Łukasz Mielańczyk, Natalia Matysiak, Olesya Klymenko, and Romuald Wojnicz. Transmission Electron Microscopy of Biological Samples. In *The Transmission Electron Microscope - Theory and Applications*. InTech, sep 2015.

[19] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation. *arXiv preprint arXiv:1606.04797*, pages 1–11, 2016.

[20] Jeffrey P Nguyen, Frederick B Shipley, Ashley N Linder, George S Plummer, Mochi Liu, Sagar U Setru, Joshua W Shaevitz, and Andrew M Leifer. Whole-brain calcium imaging with cellular resolution in freely behaving Caenorhabditis elegans. *Proceedings of the National Academy of Sciences of the United States of America*, (9):33, 2015.

[21] Seung Wook Oh, Julie A. Harris, Lydia Ng, Brent Winslow, Nicholas Cain, Stefan Mihalas, Quanxin Wang, Chris Lau, Leonard Kuan, Alex M. Henry, Marty T. Mortrud, Benjamin Ouellette, Thuc Nghi Nguyen, Staci A. Sorensen, Clifford R. Slaughterbeck, Wayne Wakeman, Yang Li, David Feng, Anh Ho, Eric Nicholas, Karla E. Hirokawa, Phillip Bohn, Kevin M. Joines, Hanchuan Peng, Michael J. Hawrylycz, John W. Phillips, John G. Hohmann, Paul Wohnoutka, Charles R. Gerfen, Christof Koch, Amy Bernard, Chinh Dang, Allan R. Jones, and Hongkui Zeng. A mesoscale connectome of the mouse brain. *Nature*, 508(7495):207–214, 2014.

[22] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. may 2015.

[23] Srinivas C Turaga, Joseph F Murray, Viren Jain, Fabian Roth, Moritz Helmstaedter, Kevin Briggman, Winfried Denk, and H Sebastian Seung. Convolutional networks can learn to generate affinity graphs for image segmentation. *Neural computation*, 22(2):511–538, 2010.

[24] J G White, E. Southgate, J N Thomson, and S. Brenner. The structure of the nervous system of the nematode Caenorhabditis elegans. *Philosophical Transactions of the Royal Society of London*, 314(1165):1–340, 1986.