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 task of performing 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 either be manually corrected, or 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: representation processing, and realignment. Specifically, we explore deep-learning based approaches that maximize representation processing 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

I acknowledge myself as author of this.

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 types 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

In the field of computational neuroscience, there exist a class of problems relating to mapping the neuron-level structure of the brain. For instance, one might want to precisely model the neural connectivity of an abnormal mouse brain, or observe the connectivity and topology of a worm at different stages of its development. 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.

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 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 submitted their predictions to several of these open competitions, often performing well.

1.3 Related Work

TODO: Flesh out more background on CNNs **TODO**: Semantic Segmentation

1.3.1 Connectomics

1.3.2 Image Segmentation

1.3.3 EM Segmentation

TODO: VD2D, VD2D-3D **TODO**: N4 **TODO**: Res Net **TODO**: Segnet **TODO**:

1.3.4 Alignment

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* [7]. 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. 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 [6].

In terms of the computational approaches to automating connectome reconstruction, many groups have attempted to create systems to accomplish near-human accuracy when labelling neurons in the brain. 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 image slices of a brain, resulting in the creation of models with near-human accuracy [3]. More recently, researchers at Princeton University

applied similar approaches to automatically detecting neurons in videos formed with the calcium imaging techniques mentioned above [1].

The work in the field continues to improve mapping techniques, but our understanding of learning methods in both computational neuroscience and computer vision are in their infancy, and are not yet practical for deployment on a larger organism (i.e. a human) due to computational and accuracy issues. This project hopes to improve on existing methods, even if incrementally, so that we can better understand what an efficient mapping approach capable of mapping a full brain might look like.

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:

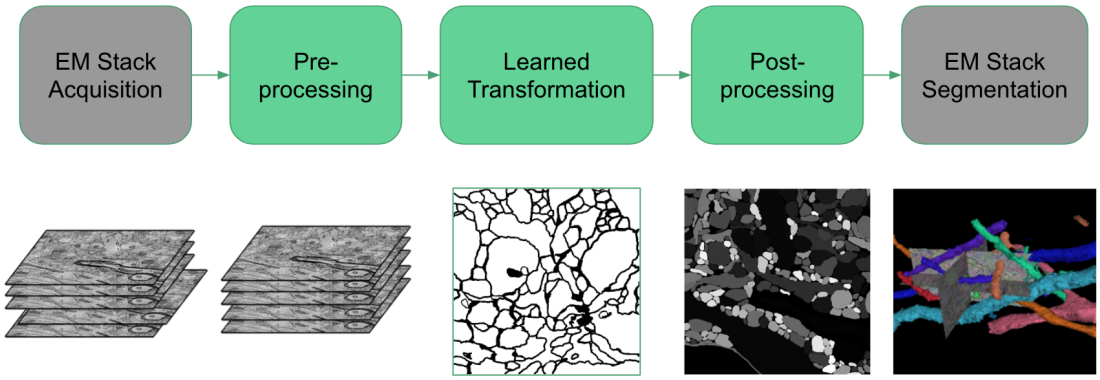

Figure 1.1: A general outline of the EM segmentation pipeline. The lower set of images represent intermediate stages that the data takes on during processing. In this example, the images are first acquired through some sort of electron microscopy technique (typically ssTEM). Second, they undergo preprocessing, 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). **TODO**: Find a citation for this 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 struc-

tures, 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 SEM. 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

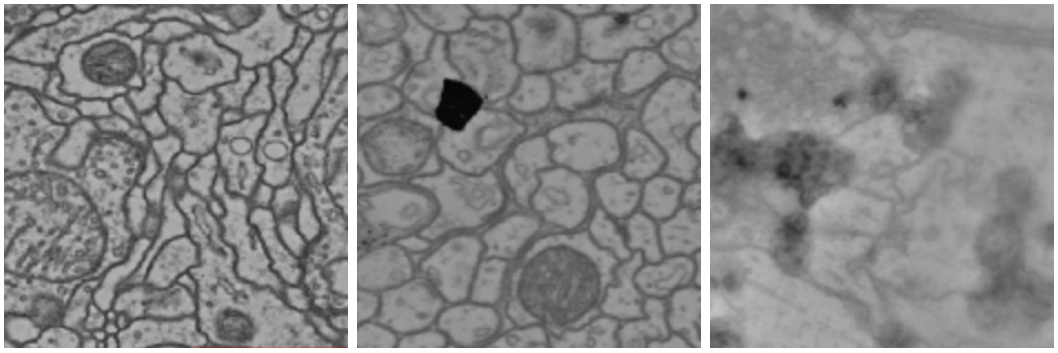

Figure 1.2: Examples of defects in the imaging process. Left: a properly stained and imaged segment. Center: a slice where inconsistent staining or another artifact has left a large dark spot on the image. Right: a slice that was prepared in such a way that the microscope couldn't produce a sharp image,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

The Seung Lab's current realignment techniques attempt to use domain-specific techniques to achieve correspondence. These techniques are typically hand-designed

[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.

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 unfeasable). 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,

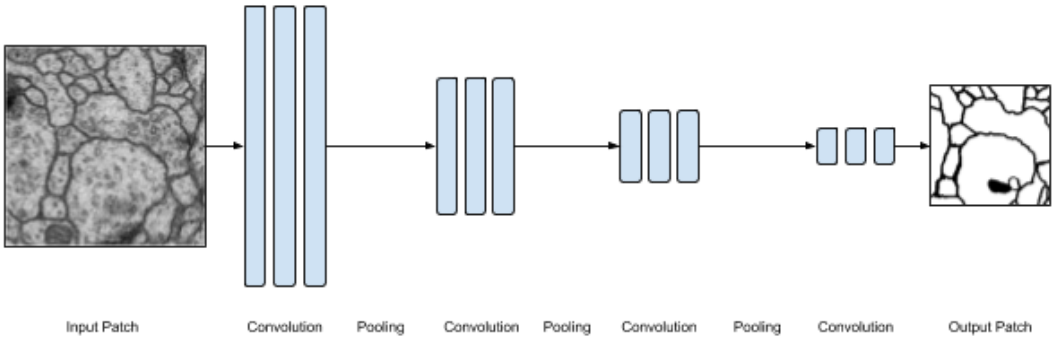

Figure 1.3: A prototypical Fully Convolutional Network, with successive convolutional layers followed by pooling layers. Depending on the types of convolution layers, the size of the input may be much larger than the size of the output, implying that the field of view for each pixel in the output could 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 som 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 will be 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.[4] 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 unrool 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).

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

[4]See Appendix B for more details about convolution.

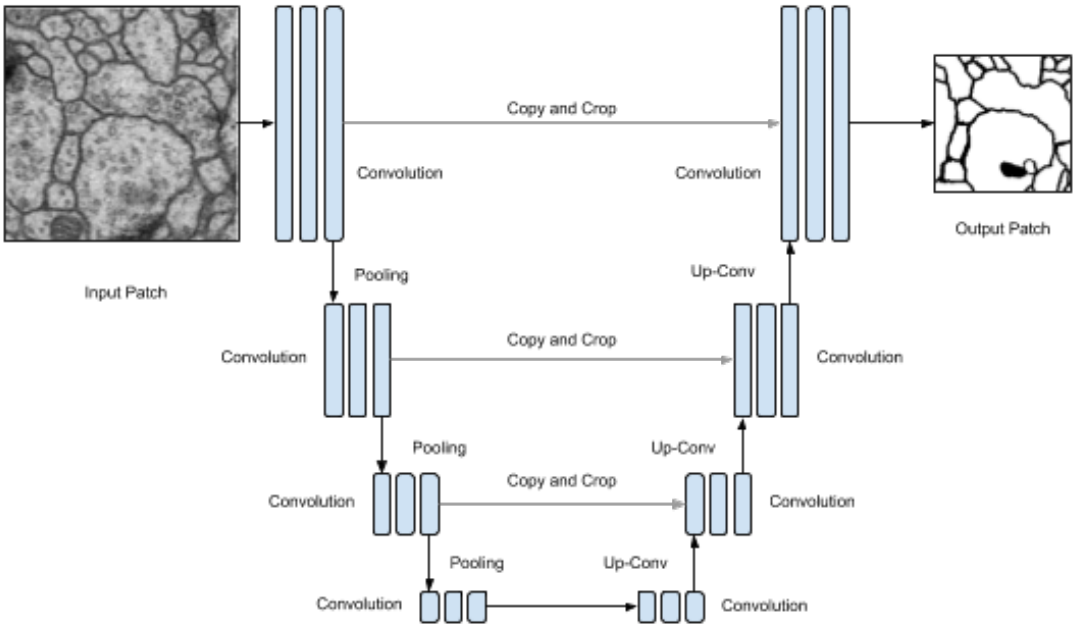

Figure 1.4: A prototypical U-Net, with successive convolutional layers followed by pooling layers on the downward pass, and then successive convolutional layers followed by up-convolutions on the upward pass. Notice how the inputs to each convolution after an up-convolution also takes as input the output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

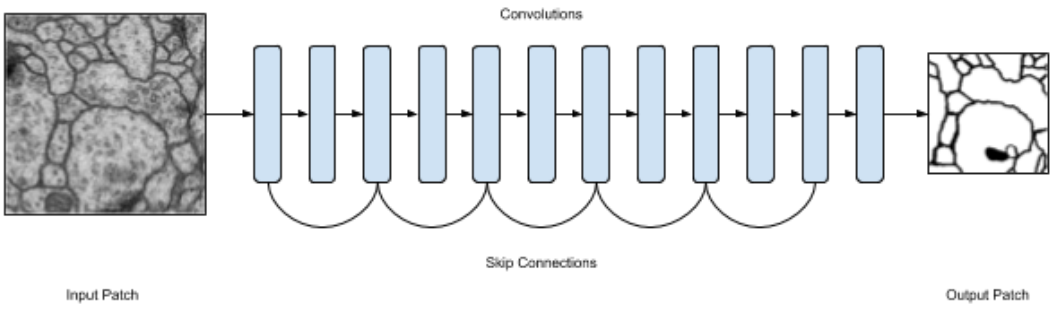

Figure 1.5: A prototypical Residual Net, which has a similar structure to the Fully Convolutional Network. The major difference is the skip connections - the residual connections - are added right before the nonlinearity but after a convolution. Residual connections can be added to other architectures 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) and 3D U-Net (3 dimensional). SegNet also resembles the U-Net architecture, in drawing inspiration from the autoencoder paradigm.

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. 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. 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, the 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. 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.

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.

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

[1] Any dataset that fits in RAM.

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

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,
4     layers=[
5         Conv2DLayer(filter_size=4, n_feature_maps=48, activation_fn=tf.
    nn.relu, is_valid=True),
6         Pool2DLayer(filter_size=2),
7         Conv2DLayer(filter_size=5, n_feature_maps=48, activation_fn=tf.
    nn.relu, is_valid=True),
```

```
8           Pool2DLayer(filter_size=2),
9           Conv2DLayer(filter_size=4, n_feature_maps=48, activation_fn=tf.
    nn.relu, is_valid=True),
10          Pool2DLayer(filter_size=2),
11          Conv2DLayer(filter_size=4, n_feature_maps=48, activation_fn=tf.
    nn.relu, is_valid=True),
12          Pool2DLayer(filter_size=2),
13          Conv2DLayer(filter_size=3, n_feature_maps=200, activation_fn=tf
    .nn.relu, is_valid=True),
14          Conv2DLayer(filter_size=1, n_feature_maps=1, is_valid=True),
15      ]
16  )
```

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.

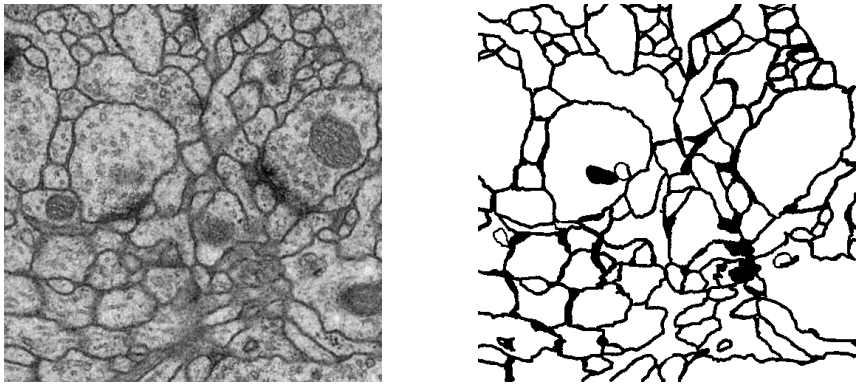

Figure 3.1: An example of 2D boundary detection. Left: the original image taken with an electron microscope. This particular example is neuron tissue taken from *Droposphila melanogaster* in a dataset created for the ISBI 2012 EM segmentation challenge [3]. The resolution of each pixel is 4nm x 4nm. 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 Error and Pixel Error. Formal definitions of both of these error metrics can be found in Appendix A.

- **Rand Error**: We will use the Rand Error to determine whether or not the segmentation process correctly labels different cells as different objects. We will also look at the Rand Split Error and the Rand Merge Error, 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**
- **VD2D**

3.4 Dataset

TODO: Talk about splits

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.

3.5 Training

3.6 Results

3.7 Discussion

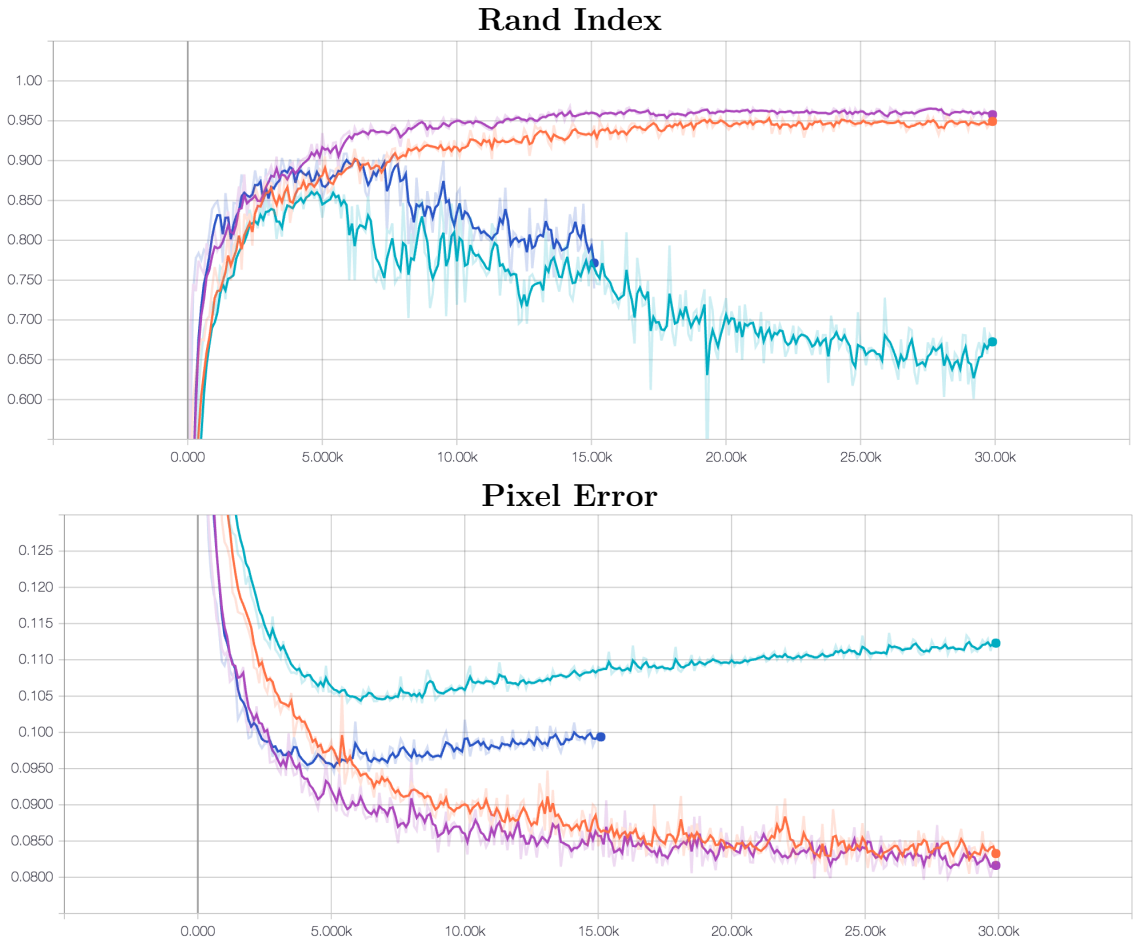

Figure 3.2: Training curves, smoothed, for 2D segmentation. Top: The full Rand scores on the validation set (from top: VD2D, N4, VD2D w/o augmentation, N4 w/o augmentation). Bottom: The pixel error on the validation set (from top: N4 w/o augmentation, VD2D w/o augmentation, N4, VD2D).

	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.

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

In recent years, the sub-problem of automatic 3D Segmentation has emerged as a popular research area. This sub-problem 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 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.

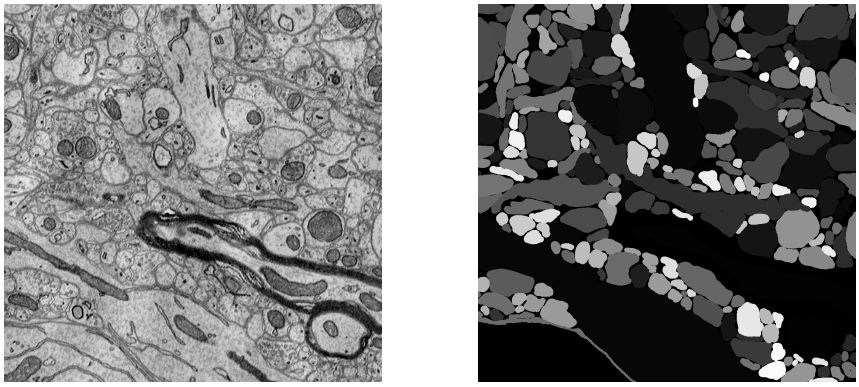

Figure 4.1: An example of a 2D cross-section of a 3D segmentation. Left: one of the original images in a stack of images taken with an electron microscope. This particular example is neuron tissue taken from the common mouse in a dataset used in the ISBI 2013 EM segmentation challenge [5]. The resolution 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.

[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.

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 Error and Pixel Error. Formal definitions of both of these error metrics can be found in Appendix A.

- **Rand Error**: We will use the Rand Error to determine whether or not the segmentation process correctly labels different cells as different objects. We will also look at the Rand Split Error and the Rand Merge Error, 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

4.4 Dataset

For our experiments

The SNEMI3D Segmentation Challenge is a highly active 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. **TODO**: Discuss our submissions to this leaderboard.

The Circuit Reconstruction from Electron Microscopy Images (CREMI) Challenge is a somewhat less-active challenge organized in advance of MICCAI 2016[4]. 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. **TODO**: Discuss our submissions to this leaderboard.

4.5 Training

TODO: Talk about splits

4.6 Results

4.7 Discussion

Chapter 5

Alignment

5.1 Task Definition

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.

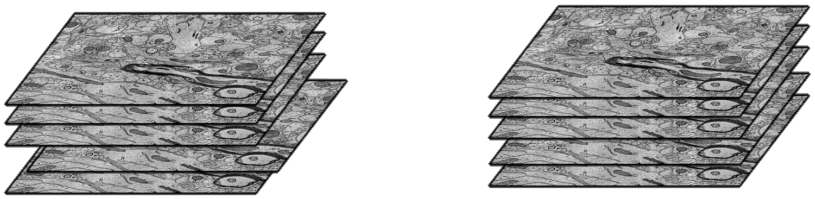

Figure 5.1: An example of a 3D stack of EM images that contains a misalignment. Left: The provided alignment of a stack. This represents a misalignment where the fourth image in a stack of images actually represents a slice slightly translated in position. Right: The correct alignment of the stack, where 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.

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 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.

5.2 Evaluation Metrics

5.3 Models

5.4 Dataset

5.5 Training

5.6 Results

Chapter 6

Conclusion

Appendix A

Metric Definitions

A.1 Rand Error

A.2 Cross Correlation

A.2.1 Smoothed Version

A.3 Pixel Error

Appendix B

Theoretical Background

B.1 Convolutional Neural Nets

B.2 Optimization

Bibliography

[1] Noah J. Apthorpe, Alexander J. Riordan, Rob E. Aguilar, Jan Homann, Yi Gu, David W. Tank, and H. Sebastian Seung. Automatic Neuron Detection in Calcium Imaging Data Using Convolutional Networks. jun 2016.

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

[5] 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.

[6] 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.

[7] 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.