Deep learning for large scale connectomic reconstruction from multibeam scanning electron microscopy data

A dissertation presented by
Hamza Kebiri

Neuroinformatics department
attached to the Institute of Cognitive Science

Under the supervision of
Prof. Dr. *Moritz Helmstaedter*
Prof. Dr. *Gordon Pipa*

In partial fulfillment of the requirements
for the degree of Master of Science in Cognitive Science

at
Connectomics Department
Max Planck Institute for Brain Research
with the collaboration of
Osnabrueck University

Frankfurt, Germany
December 2018

This work is dedicated to my parents Ali Kebiri & Malika Aboutalib
All my past, present and future accomplishments are possible due to their unconditional love and sacrifice

Acknowledgements

I would like to thank Prof. Dr. Moritz Helmstaedter for giving me the opportunity to work on this exciting project and for providing me his guidance and supervision. I am also thankful to Prof. Dr. Gordon Pipa for his co-supervision.
I am immensely grateful to Alessandro Motta for his constant help, insights and supervision during the course of my project. I would also like to thank Benedikt Staffler for the valuable deep learning discussions and for providing me the access to his code repository. I am grateful to Dr. Manuel Berning for his expertise in EM data segmentation and the many fruitful discussions I had with him. I would like to extend my gratitude to Meike Schurr for providing me with the dataset, which formed the basis of my project and Anjali Gour for biological insights and her *grandeur de coeur*. I am thankful to my colleagues Sahil Loomba and Martin Schmidt for their inputs and discussions, Sylvia Kraus-Fernando and Anna Rushing-Jungeilges for their help with all of their organizational efforts. Additionally, I would like to thank all of my lab members for providing me with a productive and encouraging working atmosphere.

Abstract

Nerve cells in the brain are densely packed together having more than a million synapses providing the connections between them. The ability to resolve such a dense and complex network at the level of single neurite and synapses has been the goal of high-resolution EM connectomics. Segmentation of data is an important step to reconstruct every single cell in a fully automated paradigm. Here, we introduce a machine learning framework based on a 3D convolutional neural network, U-Net, and a log-distance transform watershed to segment the anisotropic 3D-EM data acquired from Zeiss multibeam SEM. We used this framework to segment a 5x5x4.2 cubic micrometer tissue block from primary somatosensory cortex of mouse. The framework is successfully able to segment the data with reasonable errors. Thus, this can be further extended for scaling up the segmentation of datasets for larger volumes. This would provide a significant boost to the automated analyses and reconstruction of high resolution EM connectomics and help in gaining deeper insights into the functioning of the neuronal circuits.

Key words: *Connectomics, Machine learning, Convolutional neural networks, Image segmentation, Electron microscopy, Large scale reconstruction*

Contents

Acronyms

ANN	Artificial neural network
ATUM	Automated tape collection ultramicrotomy (Schalek et al., 2011 [14])
CE	Cross entropy
CNN	Convolutional neural network (LeCun et al., 1989 [18])
CPU	Central processing unit
DT	Distance transform
EM	Electron microscopy
FOV	Field of view
GT	Ground truth
GPU	Graphical processing unit
IED	Inter error distance (Berning et al., 2015 [22])
IMD	Inter merger distance (Berning et al., 2015 [22])
ISD	Inter split distance (Berning et al., 2015 [22])
LR	Learning rate
MCM	Mitochondria close to membrane
ML	Machine learning
MSE	Mean square error
mSEM	Multibeam scanning electron microscopy (Eberle et al., 2015 [16])
RMSProp	Root mean square propagation (Hinton et al., 2012 [41])
RNN	Recurrent neural network
SBEM	Serial blockface electron microscopy (Denk and Horstmann, 2004 [9])
SegEM	Semi-automated reconstruction pipeline for 3D-EM (Berning et al., 2015 [22])
SegEM (SF_d)	SegEM small filters deep
SegEM (LF)	SegEM large filters
SEM	Scanning electron microscopy
SEP	Squared entropy penalty
SynEM	Automated synapse detection for 3D-EM (Staffler et al., 2017 [28])

List of Figures

List of Tables

1 Introduction

1.1 Connectomics

The brain can be described from many perspectives, spanning the most detailed biological representation using genes and proteins, passing through the algorithms it is implementing, to its high level emergent properties such as consciousness. In this expanding repertoire of approaches (Figure 1.1), the field of *connectomics* aims at comprehensively reconstructing the network connections of the brain.

A *connectome*, or the connection matrix of the brain as originally termed by (Sporns et al., 2005 [2]), can potentially be informative per se, by for instance examining the circuit variations between different species or within the same specie between a healthy and a pathological brain. It can also be informative to almost all other disciplines sharing the brain goal understanding (Figure 1.1). For instance, it can help select between different neurocomputational models or rule out some of them. Connectomics goals would not have been possible without the previous work of two neuroanatomists, Santiago Ramon y Cajal and Camillo Golgi, who more than a century ago, postulated the *neurone doctrine* that comprehends the nervous system as a network of individual cells. (Cajal, 1888 [3] and Golgi, 1873 [4]). The connections between these different nerve cells or neurons are called synapses. Synapses are mainly of two types: electrical and chemical. Electrical synapses are formed between neighboring cells or neurons and are mediated via specialized membrane structures called gap junctions which are formed by channels facilitating the flow of ions and electrical signals from one cell to the other. Chemical synapses can be formed between neighboring or distant long range cells. A single chemical synapse is unidirectional, as against an electrical synapse which can be bidirectional. Chemical synapse is marked by a distinct presynaptic and postsynaptic partner. The flow of information is from the presynaptic to postsynaptic partner and is mediated via neurotransmitters. A single neuron has up to 10000 synaptic partners which thereby connect it to different neighboring and long range cortical areas.

These connections can be mapped in different scales. In the millimeter scale, the brain is split into several parcels based on either function or structure, and each of the connections

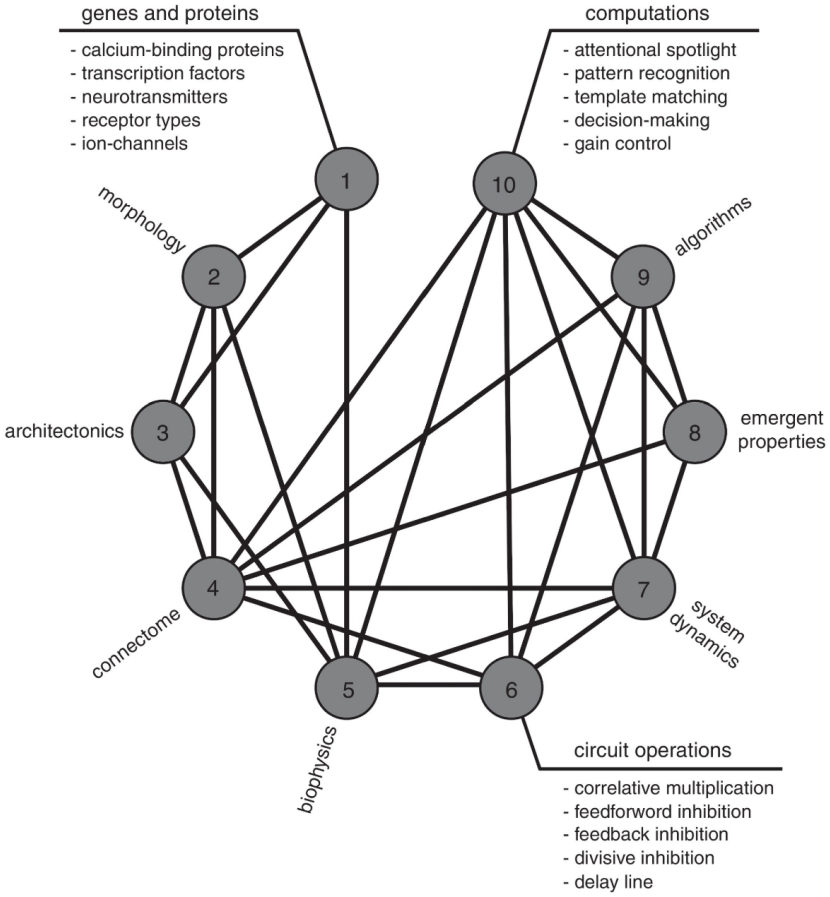


Figure 1.1 – Different perspectives to approach neural systems and their interactions [1]

between these regions are highlighted. In the mesoscale, which corresponds to hundreds of micrometers, the goal is to capture connections between neuronal populations, typically hundreds to thousands of individual neurons. In the micrometer scale, the information about each junction in every pair of neurons in the brain tissue is required. The context of this work falls into this latter category. The mesoscale studies predominantly rely on BOLD, fMRI and dTI imaging techniques, where the blood oxygenation level- dependent diffusion and the diffusion of water molecules is correlated to the activity of the respective brain regions and the diffusion patterns thereby indicating the trajectory of the different connections in the brain. Macro and mesoscale studies have also exploited the use of viral vector- based tracer injections. The injections label either the retrograde or anterograde synaptic partners, and hence enabling to understand the connectivity between different brain areas. While these methods provide a large coverage of the brain volume, they come with the major caveat of providing a low resolution.

The invention of the electron microscope in 1931 accelerated the structural investigation of the nerve tissue by revealing the structure of the the synapses at the microscale connectomic level (Gray, 1959 [5]). It took more than a decade to build the first almost fully reconstructed nervous

system of a living organism (White et al., 1986 [6] and Varshney et al., 2011 [7]) have mapped all the 302 neurons and their mutual connections of the roundworm *Caenorhabditis Elegans*. This connectome helped many subsequent research studies to understand its nervous system such as the identification of a handful of neurons that respond to light (Ward A et al., 2008 [8]).

Although the electron microscope was used since a long time to analyze the cortical tissue, the development of the Serial Blockface Electron Microscope (SBEM) by Winfried Denk (Denk & Horstmann, 2004 [9]) which led to a semi-automation of data acquisition, was a major step towards acquiring larger volume of 3D EM datasets. Several studies have analyzed neural circuits based on such 3D EM data. (Briggman et al., 2011 [9]) uncovered specific wiring asymmetry that enhances direction-selectivity in mouse retinal ganglion cells. (Helmstaedter et al., 2013 [11]) discovered a new neuronal cell type by analysing the connectivity of 950 neurons in the mouse retina, in addition to subdividing a previously known type based on its connectivity. (Takemura et al., 2013 [12]) identified cell types encoding a motion detection circuit in the *Drosophila* optic medulla. (Schmidt et al., 2017 [13]) shed light on a specific synaptic architecture of presynaptic axonal trees. They found out a path-length-dependent axonal synapse sorting rule in which excitatory axons first target inhibitory neurons followed by excitatory neurons, forming a cellular feedforward inhibition circuit.

1.2 Electron microscopy and large scale mSEM

Electron microscopy can be used in its two modalities, namely transmission electron microscopy (TEM) and scanning electron microscopy (SEM). In the former, the detected electrons are those traversing the sample whereas SEM is concerned with the back-scattered or secondary electrons from the surface of the sample. Additionally, EM methods can be categorized as block face based or slice based. The block face methods involve the tissue block being sectioned and imaged simultaneously. SBEM and FIBSEM are examples of block face based methods. While SBEM involves the use of a diamond knife for serially removing a section from the tissue block and subsequently imaging the freshly exposed surface, FIBSEM uses focused ion beam to mill away the layer of tissue from the block face.

Both these methods are destructive, i.e., the section or tissue surface once imaged is either removed or milled and thus cannot be recovered or re-imaged. The slicing based methods have an advantage over the block face method as they are non-destructive. The tissue is first sliced either manually by serial sectioning or using automated approach (ATUM) (Hayworth et al., 2006 [14]). The slices are collected on the wafer or tape and these are then imaged. While manual ssTEM is still widely used, the ATUM based methods provide a significant advantage as it eliminates manual errors. With ssTEM and ATUM-SEM, it is possible to achieve an in plane resolution of 4 nm. The high in-plane resolution is very helpful for studying mammalian cortical circuits as it is known that the dimensions of structures like spine necks and certain axons have a diameter of 40-50 nm (Helmstaedter 2013, [15]). However, imaging a large volume with 4 nm in plane resolution would be more time intensive. The multibeam scanning

electron microscopy (Eberle et al., 2015 [16]) is the new generation of microscopes that allow high throughput imaging of considerably large datasets. The setup offers a substantially high multiplexity in imaging. In the mSEM, the volume of 1 cubic mm can be processed with an acquisition rate of almost 1 GHz, resulting in only 3 months as against 15 years using a single beam setup.

This can be achieved because of two modifications introduced in mSEM: (1) the use of 61 electron beams, instead of single electron beam in the SBEM technique and (2) the parallel detection of secondary electrons. As illustrated in Figure 1.2, multiple electron beams (green) are sent in a single column and the back-scattered secondary electrons (red) are collected for imaging. The different sub-images resulting each from one single beam are then merged to output the entire image. Contrary to SBEM which allows the acquisition of micrometric scale volumes, mSEM can reach volumes in the millimeter and centimeter scale in reasonable time periods. Only 1 mm^3 of mouse cerebral cortex tissue contains at least 4 kilometers of axons, 1 kilometer of dendritic shafts and 2-3 kilometers of spine necks. This opens up new challenges in large scale reconstruction.

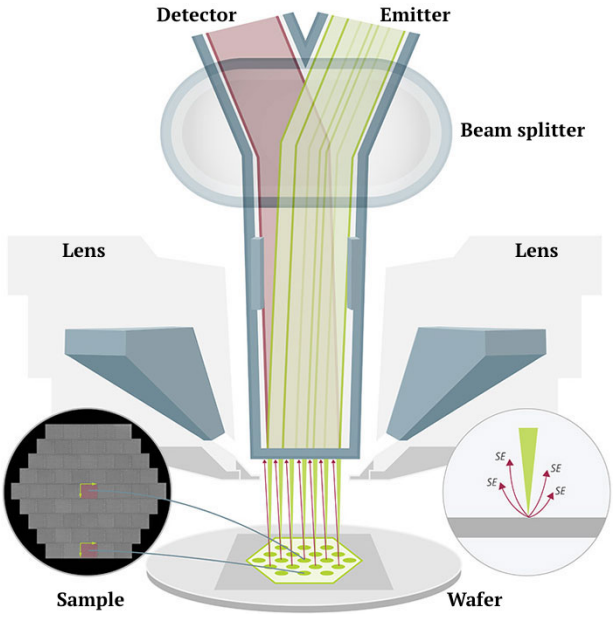


Figure 1.2 – mSEM imaging illustration [17]

1.3 Machine learning for automated reconstruction

Reconstructing the 302 neurons and 7000 synapses of the C.Elegans nervous system required approximately around 10,000-20,000 hours of manual labor. The inner plexiform layer of the mouse retina took around 30,000 hours with partial automation. Manually reconstructing a mere volume of (300μm^3) of a mouse barrel cortex would require 1 million human annotation hours, whereas the whole human cortex would take 1 billion hours (Helmstaedter et al. 2013

[15]). The need for automation is therefore not only advisable but necessary. Moreover, data acquisition by scanning electron microscopy (SEM) are generating more and larger datasets containing images of hundred gigabytes to few petabytes. Helmstaedter et al. 2013 [15] quantified the time gap between data generation and data analysis to about a three order of magnitude.

Solely using classical image processing techniques to analyse nerve tissue images has become obsolete. The current best automation procedures benefit from a subfield of Artificial Intelligence called Machine Learning. Machine learning uses specific algorithms either for classification or regression purposes. Deep Learning, a subfield of Machine Learning concerned with training deep artificial neural networks has become extremely popular since 2012. Achieving many success stories in diverse domains such as speech recognition, natural language processing and especially computer vision.

One class of deep learning algorithms inspired by the human visual system, that is benchmarked today in the computer vision domain is termed Convolutional Neural Networks (LeCun et al., 1989 [18]). They were introduced to connectomics to solve a problem in image processing called image segmentation (Turaga et al., 2009. [19] and Jain et al., 2010 [20]). It is defined as the process of subdividing an image into multiple partitions, based on characteristic features shared by voxels of each partition. Namely, in a dense connectomic reconstruction context, each partition would typically correspond to one cell.

The purpose of this work is to provide a segmentation for mSEM data, that can be built upon to detect chemical synapses that represent the basis of building a connectome. A large class of automated connectomic reconstruction approaches rely on a two step segmentation pipeline. First, a boundary prediction is performed to classify voxels into membrane or intra-cellular, followed by a post processing step to separate individual objects. The first step typically involves CNNs while the second step includes the watershed algorithm (Meyer, 1995 [30]). Another approach focuses on predicting on edges instead of nodes, by predicting affinities, i.e. how likely two voxels belong to the same object. An orthogonal segmentation approach called Flood-Filling Networks uses a single network to output individual object masks directly from raw images (Januszewski et al., 2016 [29]).

A specific CNN architecture named U-Net (Ronneberger et al., 2015 [31]), created inter alia, for EM data segmentation, is used in many recent state-of-the-art approaches ([24], [25], [26], [21]). This architecture has the specificity of bypassing the context-resolution tradeoff by including an innovative path in the network. Our work focuses on the first approach, i.e. voxel prediction mainly using different U-Net architectures, followed by a post-processing step that ends with watershed.

2 Methods

2.1 Data

2.1.1 Dataset

The dataset that is used in this work comes from the brain of a 28 day old (postnatal) mouse. It was centered in layer 2/3 of the primary somatosensory cortex. The sample tissue was squeezed by removing the extracellular space, as a result the packing density of the tissue is very high. After embedding, the tissue bloc was cut with an automated tape collecting microtome (ATUM, Hayworth et al., 2006 [14]) at 35 nm cutting thickness. Sections were collected on either Carbon nanotube tape (Kubota et al, 2018 [32]) or Carbon coated Kapton tape. Imaging was performed with a multi-beam electron microscope (61 beam setup, Zeiss, Oberkochen, Germany. Eberle et al. 2015 [16]) at 4 nm pixel size, 50 ns pixel dwell time and 1.5 kV landing energy. Experiments were performed by Meike Schurr. The size of the dataset is approximately 50 µm × 50 µm × 5 µm (Figure 2.1). Due to ATUM cutting and section collection on tape, image artefacts such as wrinkles, scratches or warping are inevitable. The segmentation task becomes even more challenging given that spine necks or axon diameters can become thinner than 40-50 nm (Helmstaedter, 2013 [15]). The slice thickness of our data is very close to that number, which means that when a small process runs parallel to the cutting plane, its continuation can be completely missed in that slice. Figure 2.2 shows the main processes that are evoked in this work, namely axons, dendrites, mitochondria, vesicles, and spine heads.

2.1.2 WebKnossos

WebKnossos (Boergens et al., 2017 [27]) is an in-browser tool used for many purposes in this work. It allows data vizualisation on the three planes spanning the 3D volume (Figure 2.3). This can be useful to explore the raw data or to qualitatively evaluate a segmentation. Training data is generated in WebKnossos by annotators through what is called *volume tracing*. This form of tracing consists of either contouring the neurite or brushing its entire surface, resulting

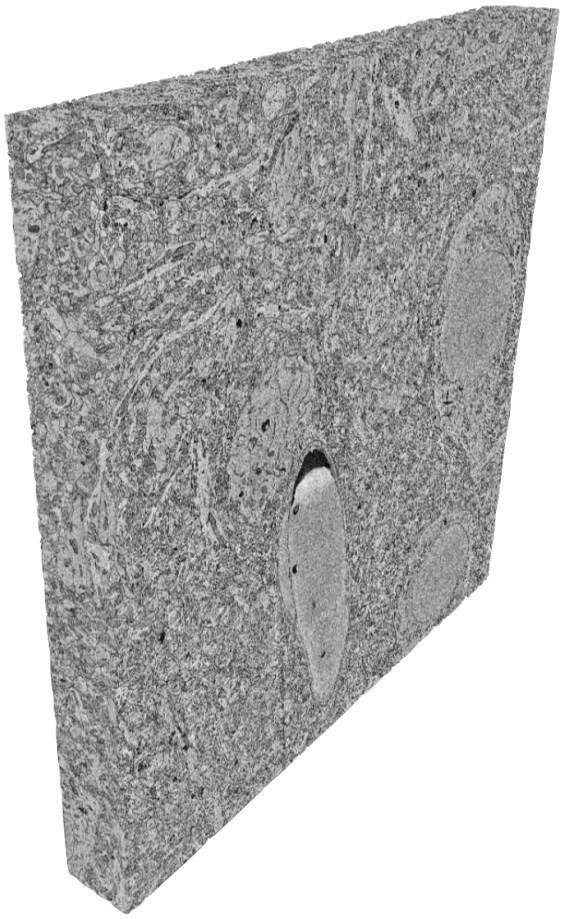


Figure 2.1 – 3D representation of the dataset

Size: ~ 50 µm × 50 µm × 5 µm
Imaging: Meike Schurr
Alignment: Ali Karimi

in a one-to-one mapping between brain cells and a unique ID that is translated to a color. In order to quantitatively evaluate our segmentations, *skeleton tracing* is also performed in WebKnossos. A skeleton, i.e. an acyclic graph containing nodes and edges, is created for each cell by placing nodes on the corresponding neurite on all sequenced slices. Skeleton tracing is also time saving to generate a volume reconstruction if given a segmentation.

2.1.3 Training and evaluation data

Originally, we have volume traced a bounding box of 2 µm × 2 µm × 4.2 µm for training and an equal one for validation. The two regions were chosen such as they contain as few voxels as possible from somata and blood vessels and as much diversity as possible by including large and small neurites. It took us around 50 hours to trace each bounding box. Since it is very expensive to generate volume training data, and in order to efficiently use the one we

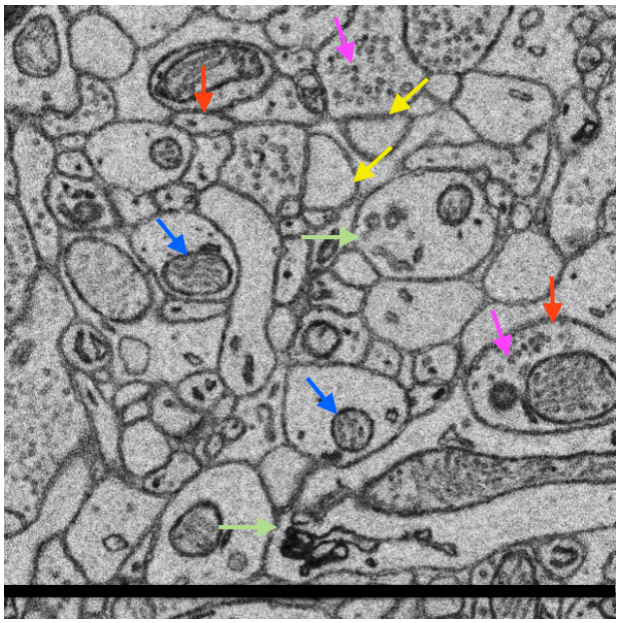


Figure 2.2 – Neuronal structures in mSEM data

Red, green, yellow, blue and purple: axons, dendrites, spine heads, mitochondria and vesicles, respectively.
Scale bar: 4 µm.

already have, it was decided to use both bounding boxes for training and assess the CNN performance visually, with the help of the training error. This approach has the drawback of not being in possession of a quantitative tool to know when should the CNN be stopped training. This can be easily checked with a validation set when the training error continues to decrease but the validation error starts to increase. We additionally traced 6 bounding boxes of 1 µm × 1 µm × 1.4 µm that contain mitochondria close to membrane. In total, our training data consists of 43 μm^3 represented by 75,000,000 voxels, that was artificially boosted as it will be described in Section 2.4.2. Note that these additional small bboxes were used from configuration 5 onwards of the U-Net (Table 2.1) and were not used for SegEM (Berning et al., 2015 [22]). Figure 2.4 shows raw and training labels from the first training bounding box. A small volume of 2.8 µm × 2.8 µm × 1.4 µm that was traced by Namrata Shettar for synapse detection was first used to evaluate the segmentation. Finally, we have skeleton traced a bounding box of 5 µm × 5 µm × 4.2 µm containing 425 neurites (Figure 2.5) with a total path length of 1.721 mm.

2.2 Machine learning & Convolutional Neural Networks

Machine learning today can be broadly split into three categories: unsupervised, supervised and reinforcement learning. Unsupervised learning infers the structure of the data solely from the data itself, supervised learning needs labels in order to build a model representing the data, while reinforcement learning builds a model by rewards or punishments. A new promising

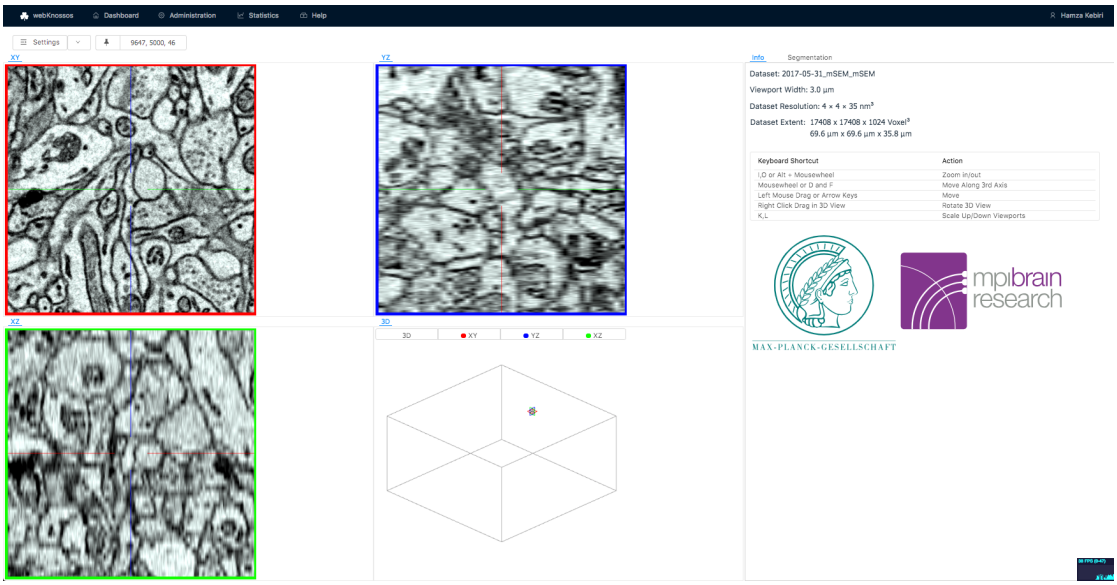


Figure 2.3 – WebKnossos interface (Boergens et al., 2017 [27])

The red, blue and green cubes correspond respectively to the XY, YZ and XZ axes.

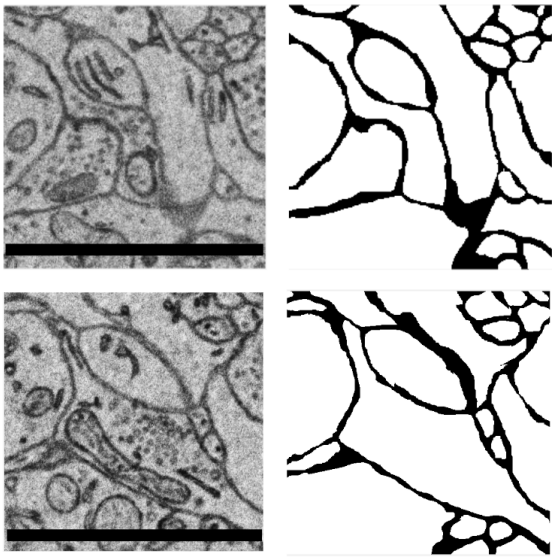


Figure 2.4 – Raw and training data

Left: raw data
Right: training labels
Scale bar: 2 µm.

emergent field of learning, that is claimed to mimic natural intelligence [LeCun, EPFL Talk, 5 October 2018] is *self-supervised learning*. It can be summarised as holes predictor where the labels are extracted from the data by the algorithm. In this work, supervised learning is used through a special class of Artificial Neural Networks called Convolutional Neural Networks.

Figure 2.5 – Skeleton trees for segmentation evaluation

Bounding box: 5 µm × 5 µm × 4.2 µm
Total path length: 1.721 mm

2.2.1 Artificial Neural Networks

Artificial Neural Networks is a broad class of algorithms running on an abstract mathematical graph in which biological neurons are reduced to single nodes and synapses to weighted edges. They are mostly used in the supervised learning domain but can also be used in an unsupervised way.

The first implementation of a model aiming to mimic natural neural networks goes back to the psychologist Rosenblatt, 1958 [33]: the single layer *perceptron*. His goal was to emulate human vision through object recognition. The structure of this neural network consists of two layers: a data or input layer and an output layer. The optimization of the network is performed through a primitive form of gradient descent. This one layer restriction as well as the non use of non linear activation functions made the field to stagnate for many years since these perceptrons could only learn linearly separable data, which usually does not represent the universe we are living in. Moving to a multilayer perceptron and to non linear activation functions made artificial neural networks handle problems that were difficult or impossible to solve by standard statistical tools.

Today, there are several types of ANNs, classified depending on the directionality of their synapses, their architecture and other characteristics. The most classical supervised learning ANN class is the Feedforward Neural Network in which the input travels in one direction by entering the input layer and the following hidden layers (if exist) and finally exiting the output layer. Contrary to a Recurrent Neural Network (RNN) and its most famous instance Long Short Term Memory (LSTM, Hochreiter and Schmidhuber, 1997 [34]) in which the output

of one layer can be fed back to the same layer. After few iterations, each neuron will form a sort memory about previous time steps. Hence this temporal weight sharing makes it the the preferred model for sequential data like sentences or stock market data.

Our class of networks of interest is called Convolutional Neural Networks (Lecun et al., 1989 [18]). In contrast to RNNs, weight sharing in CNNs is performed in the spatial domain. In fact, CNNs are very efficient in modelling data that can be hierarchically constructed from basic patterns like a visual or an auditory signal. CNNs were originally inspired from the human visual system which incorporates layering and combining inputs from low level features to high level ones. Additionally, neurons in the visual system process only a small subpart of the available information and there are many neurons which perform the same task for different parts of the visual stimulus. How CNNs perform image classification is detailed in Section 2.2.2.

2.2.2 Convolutional Neural Networks

Convolutional Neural Networks is a special instance of Artificial Neural Networks. CNNs typically comprise an alternating cascade of convolutional and pooling layers. The former layer consists of convolutions between a sliding *kernel K* and an input I, typically a raw image or some transformation of it. In the 3D context of this work, the operation in Equation 2.1 is performed for every voxel of the input I. The output of all these convolutions $conv_{K,I}$ generate a *feature map*, a transformation of the input that reflects a unique feature to be learned by the kernel.

$$conv_{K,I}(x,y,z) = \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L} K(i,j,k)I(x+i-1,y+j-1,z+k-1), \quad |K| = NxMxL \quad (2.1)$$

Since kernels are applied to different parts of the input, the same weights are learned on the whole input and not exclusively in one subpart of it. This phenomenon is known as *weight sharing*. Values of these kernels are learned through gradient descent and the *backpropagation* algorithm. In each convolutional layer, several kernels are applied to the input, resulting in the same number of feature maps. Convolutional layers are translation invariant because the convolution operator obey to this property.

A pooling layer that reduces the input dimensionality usually follows a convolutional layer. Max pooling extracts the most dominant units in a map, thereby lowering the spatial resolution while increasing the contextual information. It implicitly incorporates a mild form of position invariance since the exact position of the dominant unit is not important. Also, by reducing the input dimension, the model complexity goes down, meaning diverging from the curse of dimensionality. Pooling layers also reduce the computational cost of the model since convolutions, especially in high dimensions, are extremely expensive operators.

Typically, in a semantic segmentation context, recovering the original dimension of the input or a subpart of it is necessary in order to assign a label to each voxel. CNNs use *up convolution*

operators (also called transposed convolutions) for that aim. In fact, this step substitutes traditional upsampling algorithms like spline interpolation by making this task learnable.

2.2.3 TensorFlow framework

TensorFlow (Abadi et al., [35]) is an open source library created in late 2015 by Google Brain. While it is using Python in the front-end, the background execution is run on C++. TensorFlow aims to make machine learning faster and easier to use. In fact, additionally to implementing redundant features of the ML context like traditional libraries, it also abstracts many algorithm implementation details, ideally letting the developer focus on the main task of model understanding. Its in-browser visualization tool, *Tensorboard*, allows detailed inspection and comparison between the different trained models.
Tensorflow is based on the concept of *dataflow graphs*, in which nodes represent mathematical operations and edges the data propagating through the graph, or the *tensors*. In this work a typical graph would encompass the neural network architecture along with the optimization features and the model predictions. The data would contain both the raw images and the GT labels. When both the graph and the data are set, a *session* can be ran in which the graph get executed, and results such as weights and biases distribution, loss curve and predictions on validation images can be saved and monitored in tensorboard, in parallel to the execution. TensorFlow supports both CPU and GPU computations through a parallel computing model developed by NVIDIA named CUDA. Our CNNs are typically trained for one to two weeks in the MPG cluster, on a *Tesla V100-PCIE-16GB* GPU.

2.3 CNN architectures

This section describes the two CNN paradigms that were used in this work. SegEM (Berning et al., 2015 [22]) and two variants of it were first tested to assess how efficient it performs on mSEM data. Then, through an iterative process, a variety of 3D U-Net (Ronneberger et al., 2015 [31]) configurations, with different training strategies, were gradually set to fit the mSEM data. The U-Net architecture is implemented in the Connectomic Data Analysis Toolkit (CODAT) repository [1], a neural network framework for semantic segmentation tasks in connectomics, designed by Benedikt Staffler.

2.3.1 SegEM

SegEM (Berning et al., 2015 [22]) is the default CNN used for membrane prediction in the lab. It consists of a cascade of 5 layers, where each of the three middle layers contains 10 feature maps. The kernel is of size [11,11,5] and performs sliding valid convolutions that move with a stride of one step in each dimension at a time. The activation function that is used in all the output layers is the scaled hyperbolic tangent *tanh* [LeCun et al. 1998]. Mean square error loss

[1] https://gitlab.mpcdf.mpg.de/connectomics/codat/tree/master

between the desired output and the actual output was minimized by gradient descent using backpropagation.

Since SegEM was designed for SBEM data, and typically for a voxel size of 11 nm × 11 nm × 28 nm, it would be unfair to train it with these exact same parameters, since the physical field of view (FOV) is of a critical importance, for humans as for networks. Indeed, for an equal number of voxels in mSEM and SBEM data, there is a physical size gap factor of 6. This means that although the in-plane resolution is higher in mSEM, there will be six times less neighboring processes information than in SBEM. Hence, two other versions of SegEM were implemented to account for this FOV discrepancy. The first one named SegEM Large Filters (LF), conserves the same number of layers, i.e. 5 while increasing the kernel size to [21,21,3]. The second one, SegEM Small Filters deep (SF_d) uses kernels of size [7, 7, 3] while deepening the network to 17 layers.

2.3.2 U-Net

U-Net (Ronneberger et al., 2015 [31]) is highly predominant in almost all EM data segmentation pipelines today. Originally implemented in 2D, our work is based on a 3D one. Its architecture consists of two paths, a contracting path where the input is being shrunk by several layers of valid convolutions and max pooling, and an expansive path where a high proportion of the original resolution is recovered by up-convolutions. In addition, these two paths are bridged by combining information coming from high resolution and from high context. This innovative link (gray arrows in Figure 2.6) hacks the classical tradeoff between localisation and contextual information.

Even if we constrain ourselves to the U-Net paradigm, there is a high range of network parameters to manually choose: the number of blocks, the number of convolutional layers per block, the number of feature maps per layer, the kernel sizes in each layer, the initial input size, the pooling and the up-convolutional sizes, etc. The network input and output are interdependent and constrained by the network architecture and hence cannot be both chosen arbitrarily. For instance, if the pooling size is 2x2x2, the output coming out of all contracting path blocks has to have an even size in all dimensions. For this aim a function has been implemented to generate all the combinations of valid input and output sizes, given a maximal input dimension.

Pool and kernel sizes are two parameters to tune in order to break the anisotropy. In fact, one strategy was to avoid as much as possible downsampling in z, since the voxel size in that dimension is already high. The number of feature maps is a highly important factor since it determines the number of parameters (weights and biases) of our model and hence the model complexity. This number has to be not too big nor too small compared to the total number of voxels as detailed in section 2.4.2. The three loss functions that were tried in this work are the mean square error (MSE), cross entropy error (CE) and mean square error with entropy penalty (SEP). Several learning rates with exponential decay were tried before selecting the best one. Regarding the data, we have applied different augmentation techniques to come up

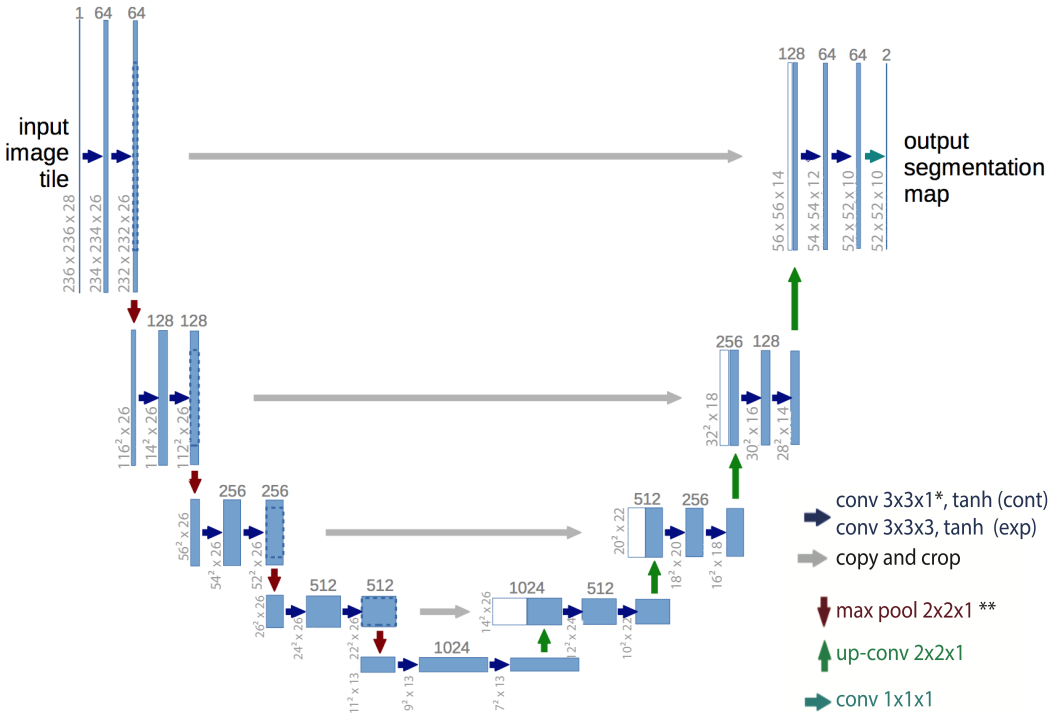


Figure 2.6 – Final U-Net architecture [31]

Used from configuration 5 to 9 (Table 2.1)

*: Except the first convolution which is 3x3x3

**: Except the last pooling which is 2x2x2

with better generalizing models. All these features are detailed in the next section. The main network configurations are shown in Table 2.1.

	Configuration 1	Configuration 2	Configuration 3
Input shape	{108, 108, 40}	{284, 284, 60}	{220, 220, 88}
Output shape	{20, 20, 4}	{100, 100, 4}	{36, 36, 16}
Kernel size	{3,3,3}	{7,7,3}	{3,3,3}
Pool size	$\{2,2,1\},\{2,2,2\}^2$	$\{2,2,1\},\{2,2,2\}^2$	$\{2,2,1\},\{2,2,2\}^2$
Feature maps	$\{2^5,2^6,2^7,2^8\}$	$\{2^5,2^6,2^7,2^8\}$	$\{2^5,2^6,2^7,2^8,2^9\}$
Loss function	MSE	MSE	MSE
Initial learning rate**	1e-5	5e-4	5e-4
Membrane penalty	1	1	1
Dropout	0	0	0
Training cubes	Sliding window	Sliding window	Sliding window
Affinities	-	-	-

	Configuration 4	Configuration 5	Configuration 6
Input shape	{108, 108, 40}	{236, 236, 28}	{236, 236, 28}
Output shape	{20, 20, 4}	{52, 52, 10}	{52, 52, 10}
Kernel size	{3,3,3}	$\{3,3,1\}^*_c,\{3,3,3\}_e$	$\{3,3,1\}^*_c,\{3,3,3\}_e$
Pool size	$\{2,2,1\},\{2,2,2\}^2$	$\{2,2,1\}^3$, {2,2,2}	$\{2,2,1\}^3$, {2,2,2}
Feature maps	$\{2^5,2^6,2^7,2^8\}$	$\{2^6,2^7,2^8,2^9,2^{10}\}$	$\{2^6,2^7,2^8,2^9,2^{10}\}$
Loss function	CE	MSE	SEP
Initial learning rate**	1e-5	1e-5	1e-5
Membrane penalty	1	4	4
Dropout	0	$\{0,0,0,0,40\%\}_c$	$\{0,0,0,0,40\%\}_c$
Training cubes	Sliding window	Sliding window	Random crop
Affinities	-	-	-

	Configuration 7	Configuration 8	Configuration 9
Input shape	{236, 236, 28}	{236, 236, 28}	{236, 236, 28}
Output shape	{52, 52, 10}	{52, 52, 10}	{52, 52, 10}
Kernel size	$\{3,3,1\}^*_c,\{3,3,3\}_e$	$\{3,3,1\}^*_c,\{3,3,3\}_e$	$\{3,3,1\}^*_c,\{3,3,3\}_e$
Pool size	$\{2,2,1\}^3$, {2,2,2}	$\{2,2,1\}^3$, {2,2,2}	$\{2,2,1\}^3$, {2,2,2}
Feature maps	$\{2^6,2^7,2^8,2^9,2^{10}\}$	$\{2^6,2^7,2^8,2^9,2^{10}\}$	$\{2^6,2^7,2^8,2^9,2^{10}\}$
Loss function	SEP	SEP	SEP
Initial learning rate**	1e-5	1e-5	1e-5
Membrane penalty	1	1	4
Dropout	$\{0,0,0,0,40\%\}_c$	$\{0,0,0,0,40\%\}_c$	$\{0,0,0,0,40\%\}_c$
Training cubes	Random crop	Random crop	Random crop
Affinities	{1,4,12} in x,y & {1,2,3} in z	{1,4,12,30,40} in x,y & {1,2,3} in z	{1,2,3} in z

Table 2.1 – Main U-Net configurations

*: Except the first convolution which is 3x3x3

**: Decay: 1% every 10,000 steps

2.4 Training procedures

2.4.1 Network parameters

Weight initialization

Ideally, we would like to initialize weights such that most of the normalized input is mapped to the undecided region of the activation function in the first forward pass of the network. For our work, since we use the *tanh*, most of the output of the network should be centred around 0, and as the network learns to distinguish between membrane and non membrane voxels, shifts towards the extremities of the function ±1.7159. We draw the initial weights w_i from the well established initialization scheme for *tanh* called Xavier initializer (Glorot & Bengio, 2010 [36]):

$$W \in [-\sqrt{\frac{6}{n_{in}+n_{out}}}, +\sqrt{\frac{6}{n_{in}+n_{out}}}] \tag{2.2}$$

n_{in} and n_{out} are respectively the number of input and output neurons of the corresponding layer.

Optimizers & Learning rate tuning

The learning rate regulates the speed of learning. A high learning rate can miss the optimal solution which will result in accuracy zigzagging, whereas a low learning rate means that the accuracy can reach an early plateau and hence the optimal solution can be missed. A good learning rate means that the error decreases steadily. Figure 2.7 shows how the error curve should behave under an appropriate learning rate.

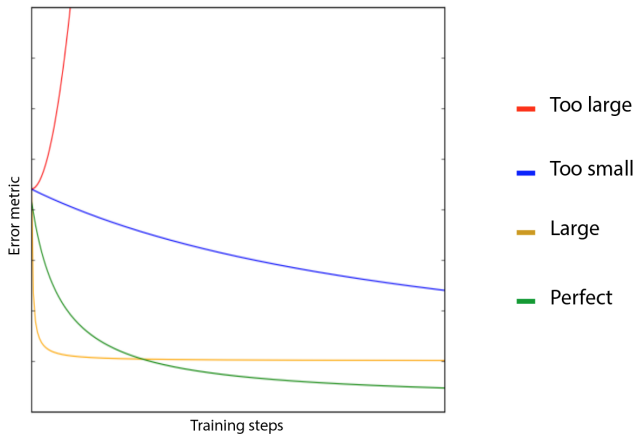


Figure 2.7 – Heuristic assessment of a good learning rate

In order to find a suitable learning rate, we have first tried high values in the range of 10^{-3}. Then small values were privileged, where we started with $5*10^{-6}$ and then we gradually increased it by factors between 2 and 10 until the training loss follows a smooth curve similar to the green one in Figure 2.7. In conjunction to this, we also monitored the training predictions to assess the suitability of the learning rate. Additionally, a high learning rate can be efficient in the early steps but can also be suboptimal when learning progress flattens out. Hence we also used learning rate scheduling where we exponentially reduce the learning by a rate of 1% every 10,000 step. In order to avoid being stuck in local minima, we add a *momentum* term of 0.9 to the parameters update.

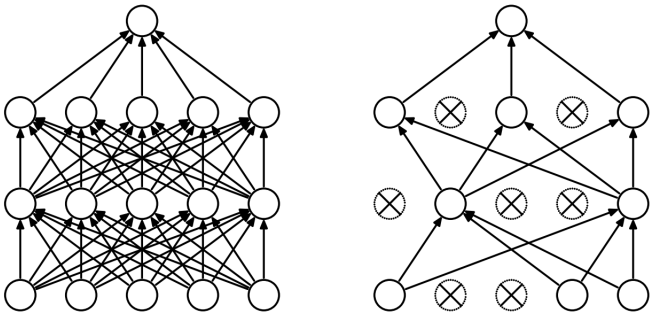
2.4.2 Regularization

Our model has to capture has to capture the class pattern of the data in order to generalize on new unseen test sets. Sometimes, the model fits the training data in an excessive manner, to the point of modelling the irrelevant details that are specific to the training set, while eventually discarding features that are important across datasets. This phenomenon is well known under the term of *overfitting*.

One configuration that may lead to this phenomenon is when the number of network parameters is too high compared to the size of the data. In contrast, a very small number of network parameters will likely result in underfitting the true distribution. There are several theoretical frameworks that try to rigorously formulate the generalization problem such as the Vapnik-Chervonenkis (VC) dimension (Vapnik, 1998 [37]) or Rademacher complexity (Bartlett & Mendelson, 2003 [38]), but their applications to deep neural networks remains unclear.

There are several ways to prevent this problem. Here we describe the solution we use in this work. The first way is to add more training data containing more diversity. But this can be expensive especially in our volume tracing paradigm. In order to get more data, one way is to artificially generate it from the training data in hand. This is known as data *augmentation*. Data augmentation in the context of image processing can be performed for instance by flipping, mirroring or rotating images. Further details are given in section 2.4.2.

Another way of preventing overfitting, that concerns the model and not the data, is to train only a randomly selected subset of the network in each training step. This can be done by dropping each neuron with a certain probability (Figure 2.8). This technique is known as *dropout* (Srivastava et al., 2014 [39]) and helps decouple neurons, hence forces patterns to be learned by different subnetworks. It can be seen as an approximation of a model averaging of the (ideally) 2^n subnetworks that were trained separately. Dropout is typically not applied in inference time.



(a) Standard Neural Net (b) After applying dropout.

Figure 2.8 – Dropout illustration (Srivastava, al. 2014 [39])

Data augmentation

Flipping and 90°rotations are the standard augmentation approaches that were used in this work. Flipping doubles the diversity of the data whereas rotations quadruples it. Combined, the resulting training data size is artificially increased eight-fold. Shifting the brightness and the contrast of images by a pseudo-random constant value was also used for some networks. A more general approach of injecting data with non-constant gaussian noise has replaced the previous approach from configuration 6 onwards (Table 2.1). This was aimed to increase the robustness and the generalization power of the network to other datasets.

Dropout

Here we use dropout (Srivastava et al., 2014 [39]) at different locations of the network and with different fraction rates. Essentially, the dropout rate represents the number of units that are set to 0. The remaining units are scaled by $\frac{1}{1-rate}$ so that the total sum remains the same at training and testing phases. We apply dropout to the output of the first block of the contractive path as well as to the output of the last block, right before the first up-convolution of the expansive path. We have applied different dropout rates ranging from 0.2 to 0.5.

2.4.3 Cost functions

MSE Voxel-wise mean square error is the first loss function that is used in this work. An optimal voxel-wise error leads to the desired segmentation. However, the desired segmentation doesn't necessarily suppose an optimal voxel-wise error, since we are more interested in the relative positioning of neurites and synapses rather than the exact labelling of each voxel in the raw data. MSE is a well established loss function for regression tasks.

MSE v.s. CE MSE supposes that the data is normally distributed whereas cross entropy supposes a binomial distribution. Although, we are performing a regression and not a classification task, we tried cross entropy as a cost function.

MSE with membrane penalty Instead of equally penalizing membrane and intra-cellular errors. We refine the loss function such that the model fosters membrane prediction. The argument is that a false predicted membrane is less dangerous than a membrane that is predicted as intra-cellular. The former may result in a split whereas the latter may end up as a merger. Different membrane penalties were tried for that aim. This can be seen as a class imbalance problem, since in a typical mSEM boundary prediction map, membrane represent around 25% of the overall voxels. Hence, in theory, a penalty of 4 is suited to our data.

MSE with entropy penalty We add an *entropy* term to the MSE loss in order for the network to predict more unsure values. In fact, the entropy function increases for undecided probabilities close to 0.5 and decreases for sure probabilities close to 0 and 1. A multiplicative parameter α is added to tune how much we want to reduce extreme probabilities. We also map the activation output to a probability to be able to use it as an entropy input.

$$SEP = MSE - \alpha * \mathscr{H} \tag{2.3}$$

With

$$\mathscr{H} = -p * log(p) - (1-p) * log(1-p)$$

$$p = \frac{(activation + 1.7159)}{2 * 1.7159}$$

$$activation \in [-1.7159, +1.7159]$$

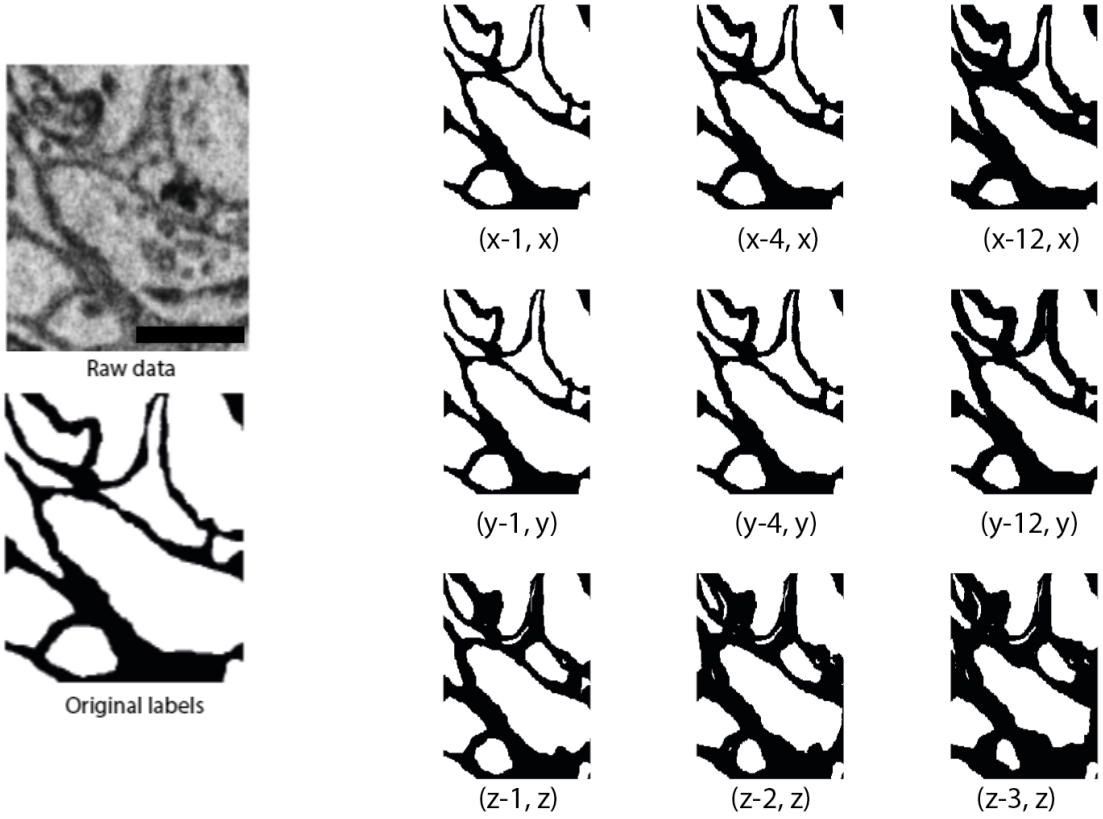
2.4.4 Cube selection

The training cubes were first selected in a sequential manner. The moving step of the sequence is the output size of the U-Net. The aim of this procedure is to decorrelate the data by feeding a strictly different input at each training step. However, this approach does not use the whole spectrum of the data because many cubes will never be seen in a complete picture by our model. Hence a second strategy was to simply randomly cropping a cube from the data. In a sufficiently high iteration number, all the configurations can be seen by our model. Random cropping is used from configuration 6 onwards (Table 2.1).

2.4.5 Range affinities

For the aim of predicting more solid and compact membranes, some networks include training with neighbouring as well as long range affinities. New training labels were generated from the original ones, by extending membranes in each dimension. Essentially, for each labeled non-membrane voxel in the ground truth map, we check along a fixed dimension, whether

there exist a membrane voxel that is proximate by less than a fixed number of voxels. If that is the case, we change the label of that non-membrane voxel to a membrane voxel. This results in several boundary maps of enhanced labels. An example is shown in Figure 2.9b, where the range affinities were chosen from two distance in-plane sets of $\{1,4,12\}$ and $\{1,4,12,30,50\}$ voxels and one z direction set of $\{1,2,3\}$ voxels (configurations 7,8 and 9 in Table 2.1). This translates in physical size to affinities of at most 160 nm in-plane and 105 nm in z. These maps were used in training, i.e. in each training step different parameters were learned for each map.



(a) Raw and original labels

(b) Affinity labels

Figure 2.9 – Neighboring & long range affinities as an auxiliary task as described by Lee et al., 2017 [24].

Networks in configuration 8,9 and 10 (Table 2.1) will be trained on original labels as well as affinity labels. The couple *(v1, v2)* in (b) represents and edge between voxels v1 and v2.
Left: nearest neighbor affinities. Middle and right: long range affinities.
Scale bar: 500 nm

2.5 Segmentation

In order to be able to distinguish the different compartments of the membrane probability map, we use the watershed algorithm. If this algorithm is directly applied to the membrane probability map, it usually generates what is called an oversegmentation, a segmentation that contains an excessive number of splits. Hence, preprocessing the boundary map is a necessary step before applying watershed. An approach that was tried consists of upsampling the CNN predictions in the z direction, applying watershed and then downsampling the segmentation to recover the original shape. Another strategy that reveals itself very efficient is the use of the distance transform. Further details are given in this section.

2.5.1 Watershed

Watershed (Meyer, 1994 [30]) is an application that takes a grayscale image as an input and maps each pixel in it to a discrete label. The name is metaphorically inspired from geology. In fact, black pixels are considered as valleys whereas white pixels are considered as hills. Hence the brightness of a pixel corresponds to the geographical height. There exists many variants of the watershed algorithm. The one that we are using is the MATLAB watershed and works as follows: the water emerges from different markers, and continues to grow until the corresponding regions are separated by a one voxel boundary, as illustrated in Figure 2.10. The markers are typically the local minima of the image.

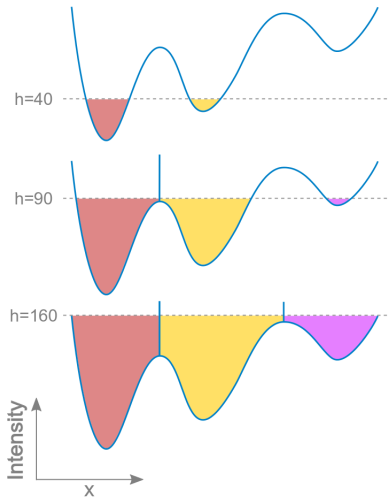


Figure 2.10 – Watershed algorithm illustration [40]

The water emerges from different local minima and fills the object until a one voxel boundary between two neighboring objects is reached.

2.5.2 Preprocessing

As mentioned in the introduction of this section, directly applying watershed will result in an oversegmentation, because of the presence of an excessive number of small local minima. The solution is therefore to get rid of these small local minima by bouncing back all the local minima whose depths are less than a certain value, that is called *h-minima* or *hmin*. This process is illustrated in Figure 2.11. Since the scaled *tanh* has its theoretical values in the interval $[-1.7, +1.7]$. The range of *hmin* has to be in this interval. After fixing *hmin*, we generate a mask that contains all the new local minima, and then we discard all connected components that are smaller than a certain value, *vmin*.

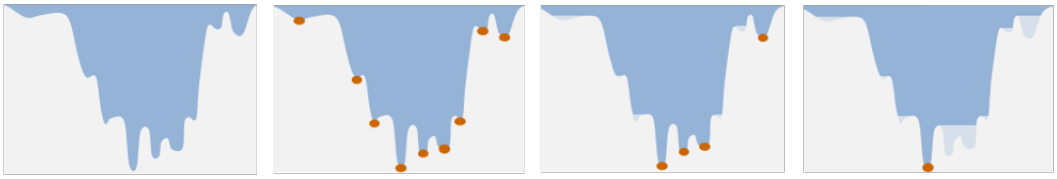


Figure 2.11 – Preprocessing step illustration

More local minima in orange are discarded from left to right by increasing the h-minima parameter.

2.5.3 Distance transform

Distance transform The distance transform is an application that takes a binary image as an input and outputs the distance between each voxel to its nearest non-zero neighbor. This can be potentially interesting to apply for our membrane probability maps after binarization, with the motivation of closing potential broken membranes.

Log distance transform The distance transform represents the height of the water when given as an input to the watershed algorithm. After applying the logarithmic function to the distance transform, all the small distances get more segregated, while larger ones get more gathered. This will encourage the watershed to further split segments that are close to the membrane, and hence avoid potential mergers that may occur because of broken membranes. To cope with the anisotropy, we replicate the different z-slices of the binarized predictions $pred_{bin}$, before feeding them to the distance transform. Also, in order for the logarithm to be correctly defined in $]0, +\infty[$, we add a one to the latter:

$$logDT = \log(1 + DT(pred_{bin})) \tag{2.4}$$

2.5.4 Resampling

The watershed algorithm supposes that input is isotropic, which is not the case for our data. An approach to solve that was to upsample the membrane predictions 9-fold in z using spline interpolation and then apply watershed. In order to recover our original dimensions, we downsample the watershed results by using a majority vote approach: in each consecutive 9 segments we select the segment ID that is the most present.

2.5.5 Hierarchical agglomeration

Merger correction is a complicated task compared to split correction. Hence we initially generate an oversegmentation in order to reduce as much as possible the number of mergers, and then we agglomerate the split segments based on the CNN prediction. We use a hierarchical agglomeration algorithm that superimpose the CNN predictions on top of the segmentation and then ranks the segment borders based on the median of the corresponding CNN probability membrane prediction. This operation is performed recursively, i.e. each time two segments are (to be) merged, the algorithm calculates the border merging scores based on the new segmentation that contains the two previously merged segments. The algorithm hence outputs a list that contains a ranking in an ascending order from the most probable two-segments to be merged to the least ones. The ideal threshold would be the one that merges as much splits as possible without inducing any merge error.

2.6 Evaluation criteria

Visually assessing the quality of a segmentation can work up to a certain extent. In this way, two segmentations can be correctly compared against a ground truth in case there is a big quality gap between the two. As soon as this gap narrows down, objective measures are necessary. Additionally, metrics allow to automatically pinpoint failure points of a segmentation. There exists many evaluation metrics to compare a segmentation to a ground truth that is typically generated by a human annotator.

The most naive metric is the *pixel error*, where the error is simply the number of pixels on which the two segmentations disagree. This metric has the drawback of rigidity: it does not tolerate small location boundary differences. The *warping error* solves this problem by focusing on preserving topological properties and hence does not punish boundary shifting at all. However, this error can penalise the presence of inner holes, which is not a desired feature in EM data segmentation.

The *rand index*, that is originally used for measuring similarity between two clusters, solves this problem by only penalising connectivity errors. This metric weights split and merge errors by their corresponding size. The average distance between splits and mergers as well as the number of splits and mergers are the main metrics of interest in this work.

2.6.1 From rand error to splits and mergers

The Rand Error is defined as 1- Rand Index. The latter quantifies how likely two segmentations would agree on a randomly chosen pair and formally follows the subsequent definition:

Given an image of n voxels, a segmentation $S = \{S_1, S_2, ..., S_m\}$ and a ground truth $G = \{G_1, G_2, ..., G_l\}$.

a : the number of voxel pairs that belong to the same segment in S and the same segment in G
b : the number of voxel pairs that belong to different segments in S and different segments in G
c : the number of voxel pairs that belong to the same segment in S and different segments in G
d : the number of voxel pairs that belong to different segments in S and the same segment in G

$$\text{Rand Index} = \frac{a+b}{a+b+c+d} \tag{2.5}$$

$$\text{Rand Error} = 1 - \text{Rand Index} = 1 - \frac{a+b}{a+b+c+d} = \frac{c+d}{a+b+c+d} = \frac{c+d}{\binom{n}{2}} \tag{2.6}$$

The last equality holds because $a+b+c+d$ represents the number of all possible voxel pairs in the image. If we closely look at this definition, we can interpret c as the sum of all merged areas in voxels and and d as the sum of all split areas in voxels.

Hence by slightly modifying the Rand Error, we can extract the number of splits and mergers from a volume tracing. The only difference resides in the fact that we don't anymore weight errors by their areas, i.e. small errors are equally contributing as large ones. This can be motivated by the argument that in the agglomeration step, a small and a large merger will have equal consequences on the final segmentation. This operation was performed by binarizing the overlap matrix between the volume traced ground truth and the segmentation. The definition of a merger and a split was further relaxed by considering an error occurrence only if the overlap is higher than a certain threshold θ. Additionally, intersections involving segment ID zero (membrane) are not included (Eq. 2.7).

This threshold was determined by visual assessment. Figure 2.12 shows an overlap of 115 in-plane voxels. This overlap is judged too small to be counted as a merger. We decided to take ~4 times this number as the limit beyond which a merger is counted. Thresholds of 400 and 500 voxels were both tried. The formal definition of the number of splits and mergers is given respectively in Eq. 2.8 and Eq. 2.9.

$$inter_{ij} = \begin{cases} 1, & |G_i \cap S_j| \geq \theta \quad \text{and} \quad G_i, S_j \neq \text{segment ID zero} \\ 0, & \text{otherwise} \end{cases} \tag{2.7}$$

$$\#splits = \sum_{i=1}^{l} max(\sum_{j=1}^{m} inter_{ij} \quad -1, 0) \tag{2.8}$$

$$\#mergers = \sum_{i=1}^{m} max(\sum_{j=1}^{l} inter_{ji} \quad -1, 0) \tag{2.9}$$

2.6.2 Split-merger rates

Based on personal experience, a skeleton tracing is 9 times faster to generate than volume tracing. Split-merger rates, as defined by Berning et al., [22], are computed based on the ground truth skeleton in one hand, and the volume segmentation in the other hand. A segment is defined to perfectly match a skeleton if and only if all its nodes are contained within that segment. Otherwise, we either have a split or a merger. The number of splits per skeleton is computed as the sum of all the different segment IDs that overlap with that skeleton. This number, minus one to account for the actual segment, is computed for each skeleton and the final number of splits n_s is the overall sum. And the number of mergers per segment is defined as the sum of the different trees that contain that segment. Similarly, this number minus one to account for the actual tree is computed for each segment and the final number of mergers n_m is the overall sum.

To be able to compare these numbers across datasets of different sizes, we normalize the number of mergers and splits over the path length L of all the neurites in the bounding box, measured in physical size. The path length can be extracted from the skeleton since, when tracing in plane, each two annotated nodes are separated by the z thickness. Hence the overall path length can be calculated as $L = \#edges * 35nm$. Our final metrics can be formulated as:

$$d_s = \frac{L}{n_m} \quad \text{and} \quad d_m = \frac{L}{n_s} \tag{2.10}$$

and the overall error is:

$$IED = \frac{1}{\frac{1}{d_s} + \frac{1}{d_m}} \tag{2.11}$$

Figure 2.12 – Merger threshold assessment in volume based split-merger metric

Left: ground truth volume and its segmentation.
Middle: mask of a segmented and a ground truth object that overlap.
Right: overlap area judged too small to be considered a merge error.

3 Results

3.1 SegEM network evaluation

Originally, we wanted to assess the performance of SegEM (Berning et al., 2015 [22]) and its variants on the mSEM data. Training in one bounding box and validating on the second one every hour shows superiority of SegEM (LF) over SegEM (SF_d) and SegEM in the training loss, while in the validation loss SegEM (SF_d) performed better over the two. (Figure 3.1). As motivated in Section 2.1.3, we have decided to train on both bounding boxes. The training error of the different versions of SegEM is shown in Figure 3.2. It took more than 3 days for SegEM (LF) to reach 8,000 iterations, whereas it took around half of that time for SegEM (SF_d) to reach the same step while achieving a better voxel-wise error, confirming the validation error trend earlier. In fact, for an equal FOV, small kernels in deep networks are usually preferred over large ones in shallow networks. Unsurprisingly, both networks outperformed the original SegEM that has a smaller FOV. Membrane probability maps of the three versions of SegEM are shown in Figure 3.3. The lower error rate of SegEM (SF_d) is, inter-alia, because it has better learned the concept of vesicle clouds and mitochondria than the two other versions. However, when mitochondria or any small structure is very proximate to the membrane, correctly predicting this structure may come at the cost of predicting intra-cellular voxels in this very proximate membrane. As a result, a membrane breakage will happen which will increase the likelihood of merging the two processes at the watershed step.

3.2 U-Net evaluation

Learning rates Initially we have tried configuration 1 (Table 2.1). The first challenge was to find a suitable initial learning rate. Figure 3.4 shows the error for several learning rates as described in section 2.4.1. We have decided to fix its value to 10^{-5}, since the convergence under $5*10^{-6}$ is too slow while the curve onset of $5*10^{-4}$ and 10^{-4} is a little bit too dramatic. We have also tried two different optimizers: RMS Prop (Hinton, 2012 [41]) and Nesterov Momentum (Sutskever et al., 2013 [42]) but the error rate was not reduced (Figure 3.5). Hence, we kept

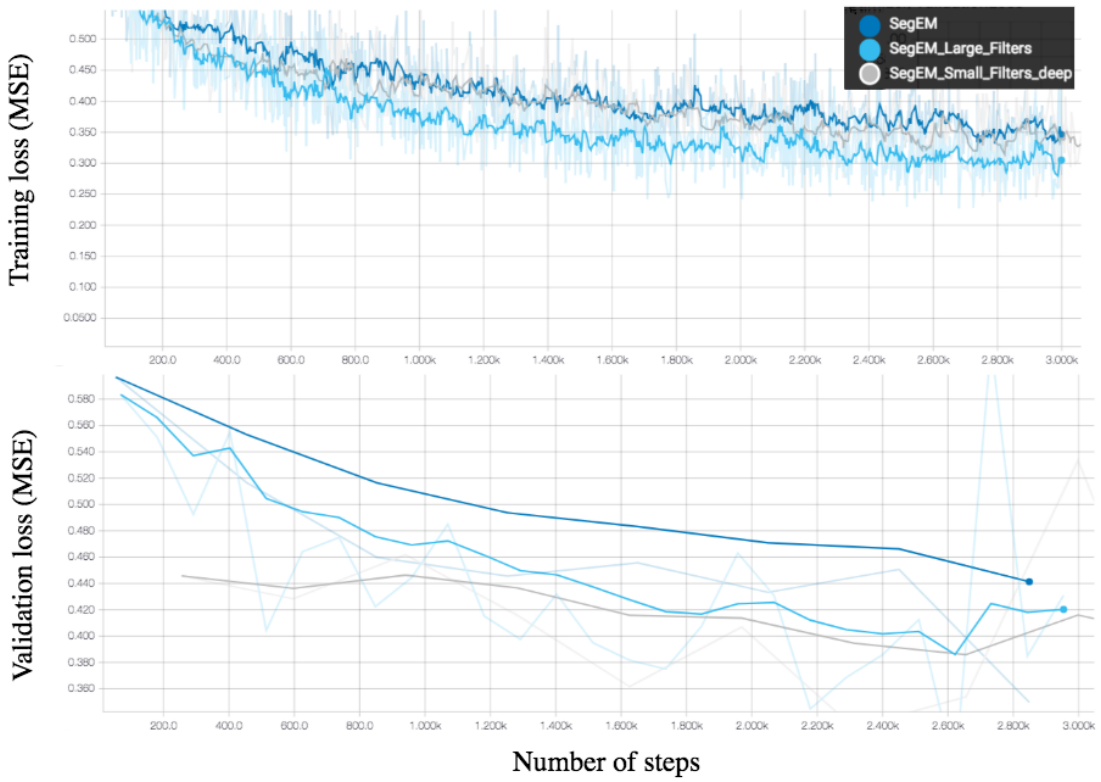


Figure 3.1 – Training and validation errors on different versions of SegEM (2.3.1) [22]

Training data: 1 bounding box of 2 µm × 2 µm × 4.2 µm
Validation data: 1 bounding box of 2 µm × 2 µm × 4.2 µm
Validation frequency: 1 per hour

using stochastic gradient descent with momentum for all subsequent configurations.

We have noticed that the error approaches ~0.08 compared to ~0.24 for SegEMs. In fact, this does not necessarily mean that the membrane prediction is three times better for the former, since by predicting more sure values, the overall error may decrease. Indeed, this is confirmed by the membrane probability map in Figure 3.7. U-Net maps output more sure probabilities than SegEM ones, both for membrane as for intra-cellular voxels. Additionally, membrane voxels are thicker in the former.

Cross-entropy We have tried cross entropy error instead of mean square error (configuration 4) with the hope to generate less binary outputs. Figure 3.6 shows that indeed the predictions are more gray. Nevertheless, we observe more broken membranes than with MSE.

More degrees of freedom We have also trained two other networks that contain more parameters. In the first one (configuration 2 in Table 2.1), we increased the kernel size from {3,3,3} to {7,7,3} and for the second one (configuration 3) we added another bloc of 512 feature maps.

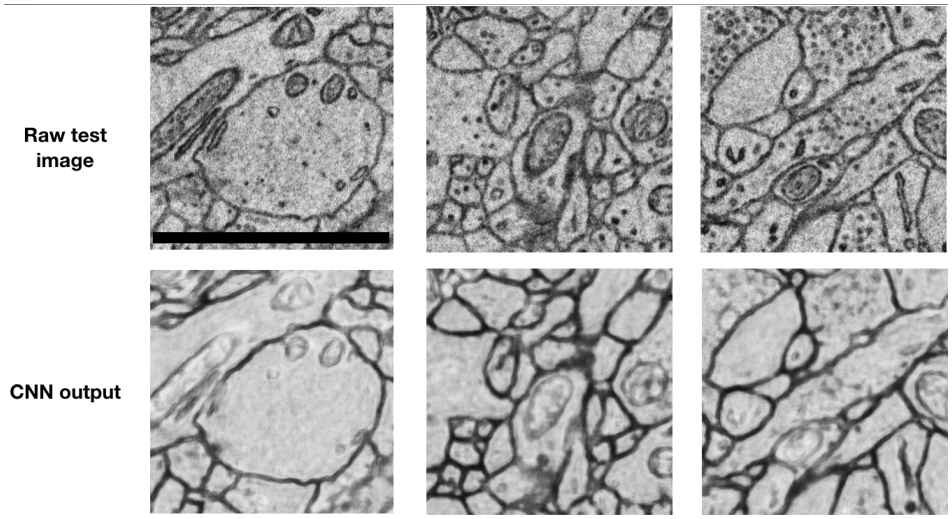
Figure 3.2 – Training error on different versions of SegEM (2.3.1) [22]

Training data: 2 bounding boxes of 2 µm × 2 µm × 4.2 µm

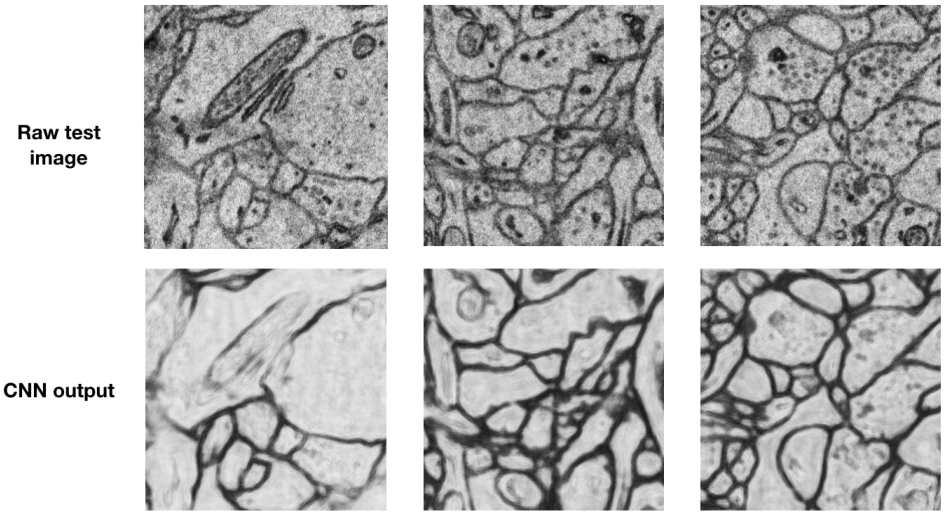
Despite a better training error rate compared to configuration 1, the training was very slow for configuration 2, and the probability map contained many broken membranes as shown in the example of Figure 3.8. Also, the network generated more binary predictions compared to configuration 1 which may be a sign of overfitting the training data. In configuration 3, the predictions were clearly binary which made the overfitting hypothesis more likely. In fact, these two networks, started to overfit very quickly, Histograms of Figure 3.9 show the predictions at ~40,000 steps are less binary than those in at ~230,000 steps. Furthermore, predictions on the training are highly accurate and very completely binary.

From this point, we have decided to keep working with small kernels. Additionally, we stopped pooling in z in the early blocs. We also increased the number of blocks and the number of feature maps. Hence, a large number of parameter which will likely result in overfitting. To counter this problem we used all the regularization techniques that are described in Section 2.4.2.

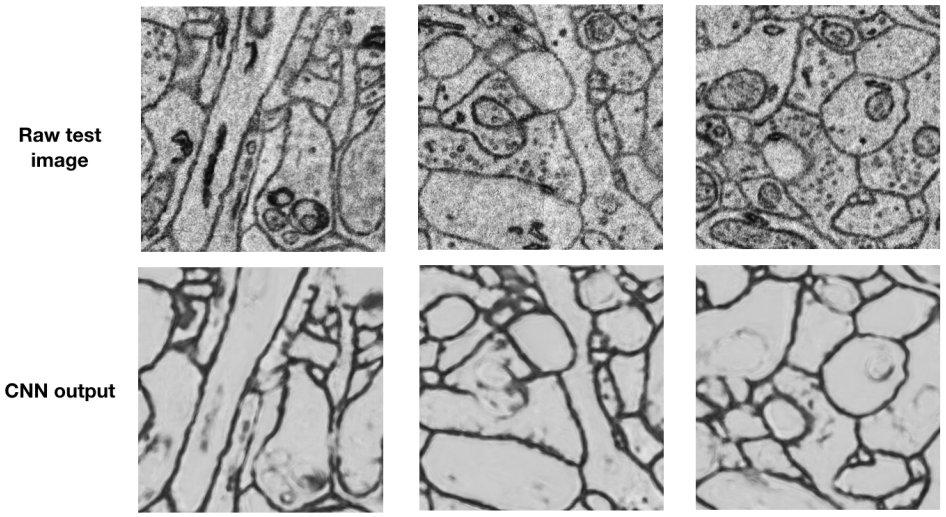
Dropout experiment The first regularization technique we used is dropout (Srivastava et al., 2014 [39]). Figure 3.10a summarizes four main configurations in which we apply the dropout rates that are specified in the bottom of each CNN output. For instance, 0.5-0-0-0.5 corresponds to a dropout rate of 50% applied to the output units of the first contracting block, no dropout in the second nor the third contracting block and a dropout rate of 50% to the fourth and last contracting block. Visually, the best configuration is the 0-0-0-40%. The training error in Figure 3.11 confirms this in the early steps but the configuration 0.5-0-0-0 shows a similar voxel error in later steps. Additionally, the idea of dropout in the first block seemed appealing since the dropout effect can cascade to other layers without explicitly applying dropout to them. Both configurations are compared in Figure 3.10b and the former is clearly performs better. The authors of the original U-Net also claim to use dropout in layers at the end of the contracting path. We then stick to this dropout configuration for all the



(a) Original SegEM [22]



(b) SegEM Large Filters



(c) SegEM Small Filters deep

Figure 3.3 – Raw data and CNN output for different SegEM versions (2.3.1) [22]

Mitochondria and vesicle clouds are better learned by SegEM (SF_d) but confused when the former or any dark structure is very close to the membrane which may potentially lead to a merger.
(Scale bar: 2 µm, applies to all subfigures)

Figure 3.4 – Training error curve under different learning rates

Running networks of configuration 1 (Table 2.1) on different initial learning rates with decay to choose the best one based on the curve shape and the training error.

upcoming networks starting from configuration 5. Figure 3.10c shows the CNN outputs of two membrane predictions at ~ 2 million steps.

Membrane penalty As a hope of reducing potential to be mergers by fostering the network to predict more membrane voxels than intra-cellular. We have trained networks with different membrane penalties. Figure 3.12, shows that the training error for a membrane penalty of 3 is lower than penalties of 7 and 12. This is not a surprising result, since there are more intra-cellular voxels that are wrongly predicted as membranes. However, the hope is that there are also more membrane voxels that are correctly predicted. We have decided to set a conservative membrane penalty of 4 which is the theoretical best class imbalance compensator (see section 2.4.3).

MSE with entropy penalty (SEP) & noise injection One major source of our network errors is when a mitochondrion is close to a membrane (MCM). As a result, this very membrane is broken as shown in Figure 3.13 (top left). Noise injection (top right) drastically fixes this issue. Mean square error with entropy penalty (SEP, configuration 6 in Table 2.1) fixed the issue partially (bottom left). Hence, driven by a conservative motivation, we have decided to combine both approaches (bottom right). Note that the this comes at the cost of predicting more false positives (membranes) but this is not a concern since the worst case result is a cheap split error.

Enhanced labels Figure 3.15 shows the membrane probability maps for the short and long range affinities (configuration 7, 8 and 9 in Table 2.1), compared to the previously best generated map of SEP with noise injection (configuration 6). Clearly, enhancing the labels with more membrane voxels in the three dimensions did not result in a better outcome. This is

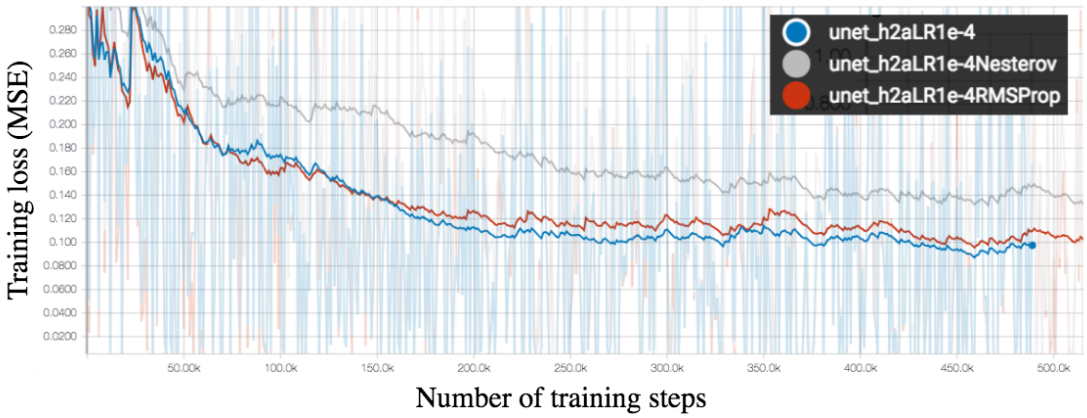


Figure 3.5 – Training error curve under different optimizers

Comparison between three optimizers for a fixes initial learning rate of 10^{-4}: Stochastic gradient descent with momentum (blue), idem with Nesterov momentum [42] (gray) and RMSProp optimizer [41] (red) with momentum.

confirmed by the significant different in training error (Figure 3.14), which likely means that additionally to the false positives (mergers to be) that are shown in Figure 3.15 there are also more false negatives (split to be) that pushed the error upward. Hence we kept using the network of the configuration 6 (Table 2.1).

3.3 Segmentation evaluation

3.3.1 Watershed

Based on volume tracing We have performed a grid search on a spectrum of hmin and vmin values, as well as the binarization thresholds for the distance transform. In total, 351 combination was tried for the former and 1539 for the latter. Figure 3.17 shows the distribution of the number of splits and mergers for two overlap error thresholds 2.6.1 with and without distance transform. Clearly, results by directly applying watershed outperform the ones with distance transform. The data points in the origin correspond to extreme cases when the segmentation is extremely oversplit (very low h-min) and we remove bigger objects than (high v-min) resulting in no segments and hence no merge nor split errors. The optimal parameters, in terms of least mergers for the error threshold of 400 voxels are {0,07,0.09,0.11,0.25,0.27,0.29} and {20} respectively for h-min and v-min. Figure 3.16 shows how the different segmentations with *hmin* in {0.11, 0.25, 0.55} and vmin of 20 voxels look like.

Based on skeleton tracing Because the watershed algorithm (Meyer et al., 1994 [30]) supposes a one-voxel boundary, which is highly violated in the anisotropic mSEM data, a "bleeding" or "leaking" effect is observed in the membrane voxels that spreads to neighbouring cells. In other words, small fragments of a segment are spread out to adjacent cells resulting in very

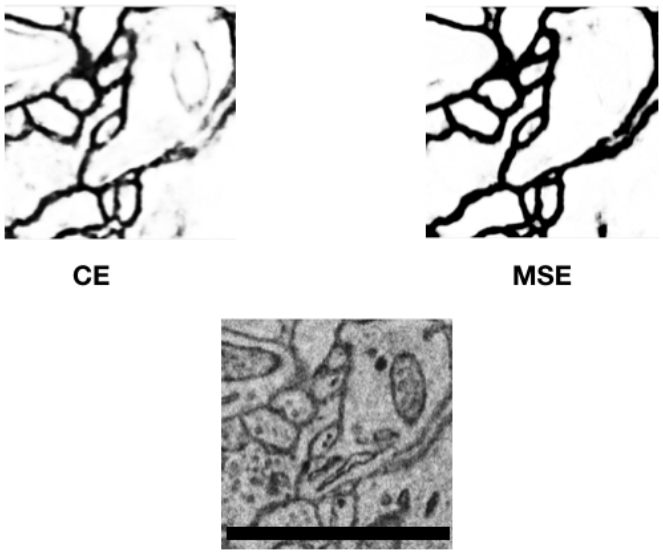


Figure 3.6 – Comparing MSE to CE loss functions

Top: membrane probability maps under cross entropy loss (left, configuration 1) v.s. mean square error (right, configuration 4). Configurations described in Table 2.1.
Bottom: raw data
Scale bar: 2 µm

small mergers. An example is illustrated in Figure 3.18.

In order for the metric to reflect what we qualitatively judge as mergers, we get rid of these small fragments by using the binarized membrane CNN map as well as the segmentation map. All the voxels that are predicted as membrane, are assigned a zero segment ID (membrane) in the new segmentation. This can eventually create new splits, where the two split segments have the same segment ID. Hence we rerun connected components and assign different segment IDs to each component.

Based on this strategy, we generate the inter-merger/inter-split distance in Figure 3.19 for three different configurations. In the first one we directly apply watershed after the preprocessing step described in Section 2.5.2. In the second one, we apply the distance transform to the CNN predictions before applying watershed, with the hope of reducing membrane breakages. And the third approache concerns the log-distance transform as detailed in Section 2.5.3.
Applying DT followed by watershed underperformed directly applying watershed as in the volume based evaluation. Also, the resampling approach described in Section 2.5.4 resulted in very poor segmentations. LogDT then watershed outperformed directly applying watershed. Hence we kept using it for all the CNN outputs.

By visually inspecting splits, we found out that most of them fall into two categories: (1) close and in membrane voxels and (2) small segments rather than average or big ones. Figure 3.20 shows a phenotype of this kind of errors.

Figure 3.7 – Membrane predictions (configuration 1, Table 2.1)

Network trained during:
Top: ~140,000 steps
Bottom: ~ 370,000 steps

3.3.2 Merger analysis

The U-Net of configuration 6 drastically solved the MCM errors as can be seen in Figure 3.21. Table 3.1 presents the ten optimal parameters in terms of mergers for the Log-DT configuration along with their respective inter-merger and -split distances.

The configuration (h-min, θ-LogDT) = (0.5, 0.7) contained exactly 10 mergers. Three of them are caused by data artefacts: two staining problems in a small process and a broken membrane. One merger is caused by a dark mitochondrion that is close to a small process (MCM). Another corresponds to a missing membrane in the raw data. Four mergers are caused by the bleeding effect that was not solved by the postprocessing step using the binarized membrane predictions. One is partially due to a misplaced node and the bleeding effect. Figure 3.22 shows the five true mergers.

3.3.3 Agglomeration

After applying the described hierarchical agglomeration algorithm in Section 2.5.5, we faced the problem that the segments in the agglomerates, even if they contain the same segment ID, are not connected components. Hence the same strategy as in 2.6.2 to compute split and merger rates cannot be directly applied. To cope with this problem, we have decided to not run connected components after assigning a segment ID zero to voxels with high membrane probability. This comes at the cost of eventually missing the detection of splits,

Figure 3.8 – Membrane predictions (configuration 2, Table 2.1)
Network training during ~150,000 steps
Too binary (overfitting symptom)

h-min	θ-LogDT	d_m (µ m)	d_s (µ m)
0.8	0.7	90.57	1.21
0.5	0.6	90.57	1
0.5	0.8	101.22	0.68
0.2	0.5	101.22	0.6
0.4	0.6	107.55	0.88
0.7	0.7	114.72	1
0.6	0.7	132.37	0.76
0.3	0.6	132.37	0.71
0.2	0.6	143.40	0.54
0.5	0.7	172.08	0.6

Table 3.1 – Best inter-merger distances and corresponding inter-split distances along with segmentation parameters

even if the likelihood of completely splitting a segment with this approach is very low, which was confirmed after visual inspection. Figure 3.23 shows the average distance between splits and mergers for different agglomeration thresholds. Decreasing the threshold towards less splits increased the inter-merger error drastically.

3.4 Application to a novel dataset

We run our segmentation pipeline on a new dataset that our networks were not trained on. This dataset was acquired from layer 4 of the the primary somatosensory cortex of a 28 day old mouse. It was imaged using the same parameters (4x4x35nm voxel size, 50ns pixel dwell time, 1.5kV landing energy). However, the sections are collected on a different support tape, carbon coated Kapton tape for this one as against CNT tape (Kubota et al., 2018 [32]) for the previous one. The size of the new dataset is ~0.8 mm^3 and goes 100 µm in z. Because of the different tape that was used, the dataset contained more noise, including missing or completely damaged slices. Contrary to the old dataset, we also observe the presence of myelin sheaths. The performance of the segmentation on this dataset, although suboptimal compared to the original one, has drastically improved after training our networks with noise injection. Generalising to novel datasets is a common problem in connectomics that is typically solved by retraining the networks to adapt to the new data distribution. Figure 3.24 shows a segmentation example on this dataset by our current pipeline.

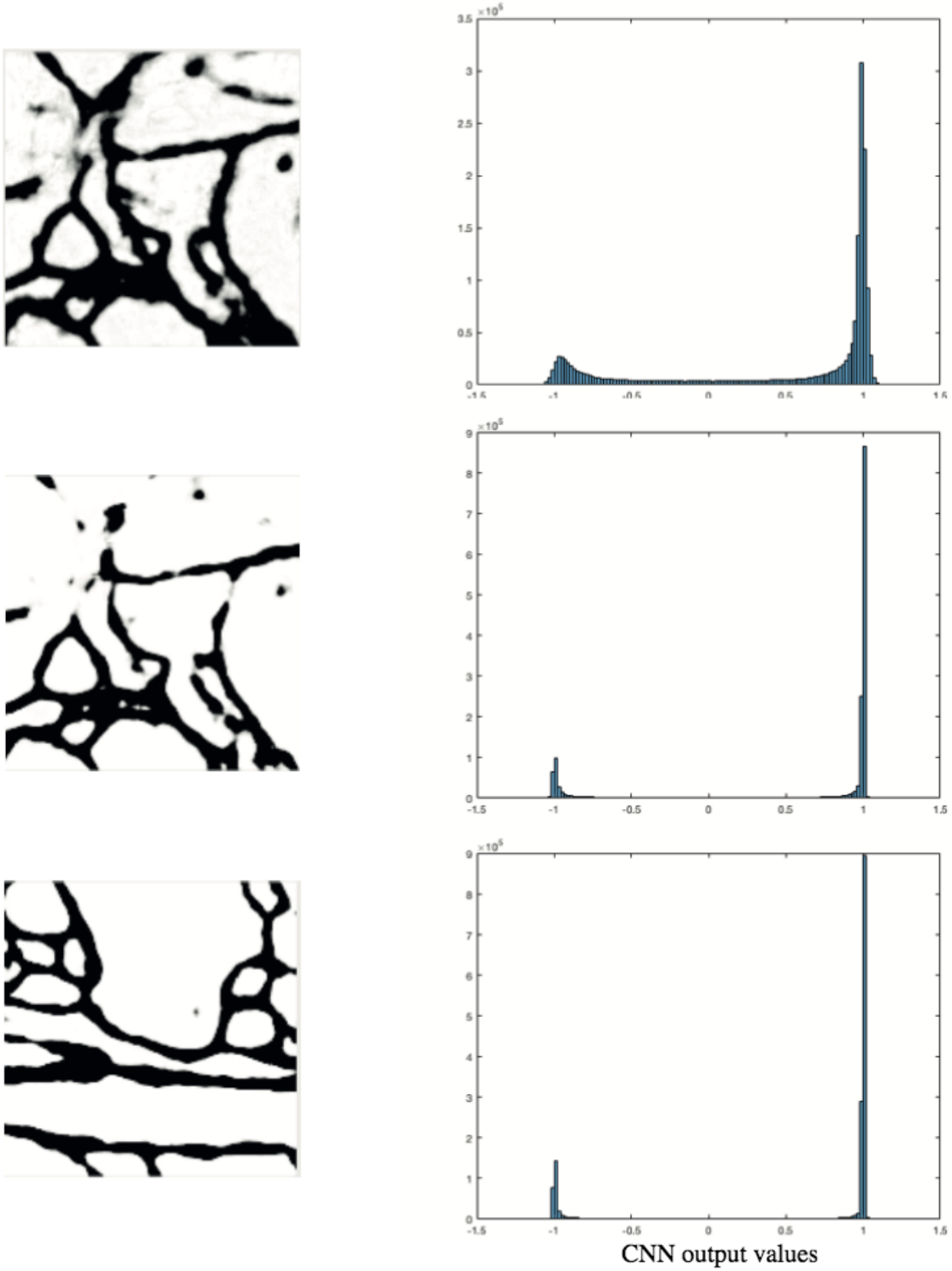
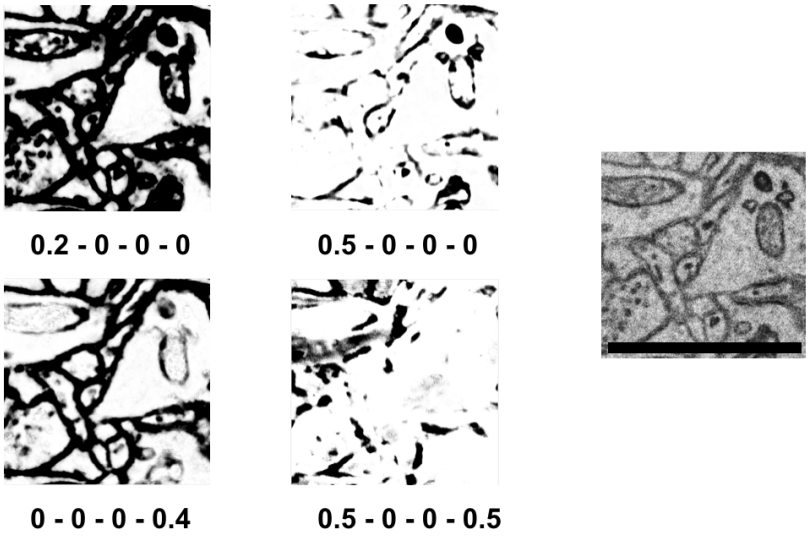
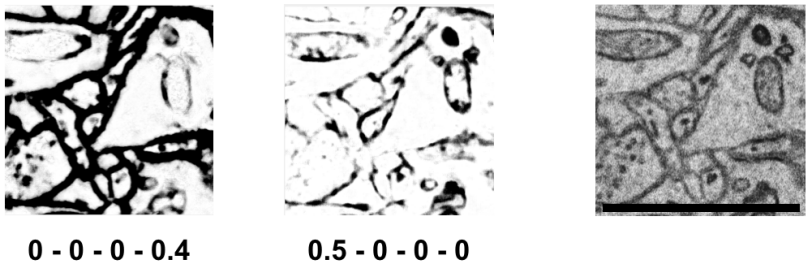


Figure 3.9 – Membrane predictions (configuration 3, Table 2.1) and corresponding histograms

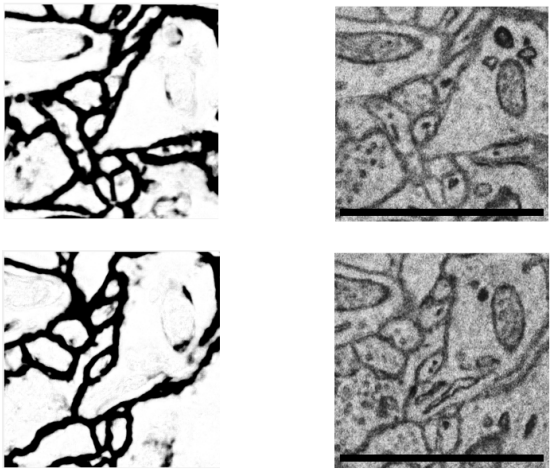
The network is clearly overfitting the training data, since the predictions are highly binary at ~230,000 steps (Middle) compared to ~40,000 steps (Top) as showed by their respective histograms. Predicting on a training cube at ~230,000 steps revealed a more binary and highly precise output, confirming the overfitting hypothesis.



(a) Different dropout configurations at ~650.000 steps



(b) Two competing dropout configurations at ~1.000.000 step



(c) Dropout fixed to output of last contractive path layer:
0-0-0-0.4 (~2.000.000 steps)

Figure 3.10 – Dropout experiment

We try several configurations in (a). The two best configurations in terms of training error are trained more. (c) CNN outputs on the best dropout configuration.
*DOX*_0_0_*Y* means we dropout *X* of the neurons from the output of the first block of the contractive path, no dropout in the two following blocks, and *Y* dropout in the output of the last blocks of the contractive path.
Scale bar: 2 µm

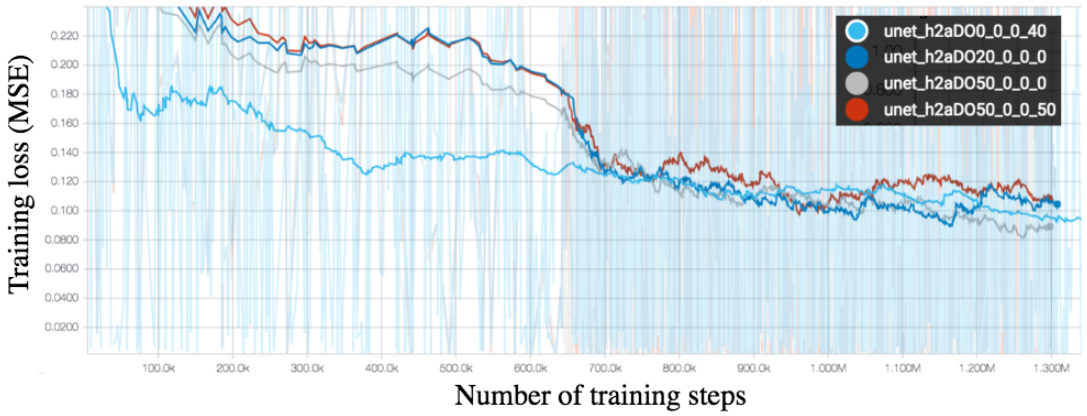


Figure 3.11 – Training error on different dropout combinations using configuration 5 (Table 2.1)

$DOX\_0\_0\_Y$ means we dropout X% of the neurons from the output of the first block of the contractive path, no dropout in the two following blocks, and Y% dropout in the output of the last blocks of the contractive path.

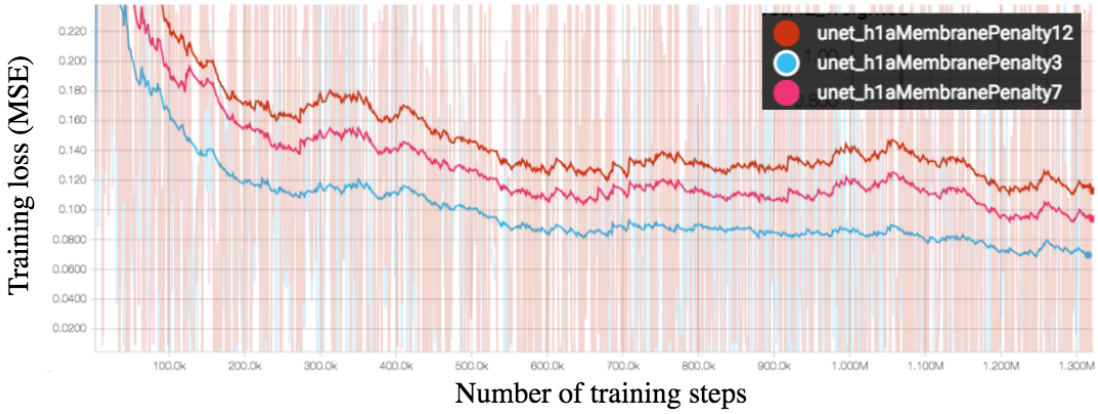


Figure 3.12 – Training error on three membrane penalties using configuration 1 (Table 2.1)

Membrane penalties: 3, 7 and 12.

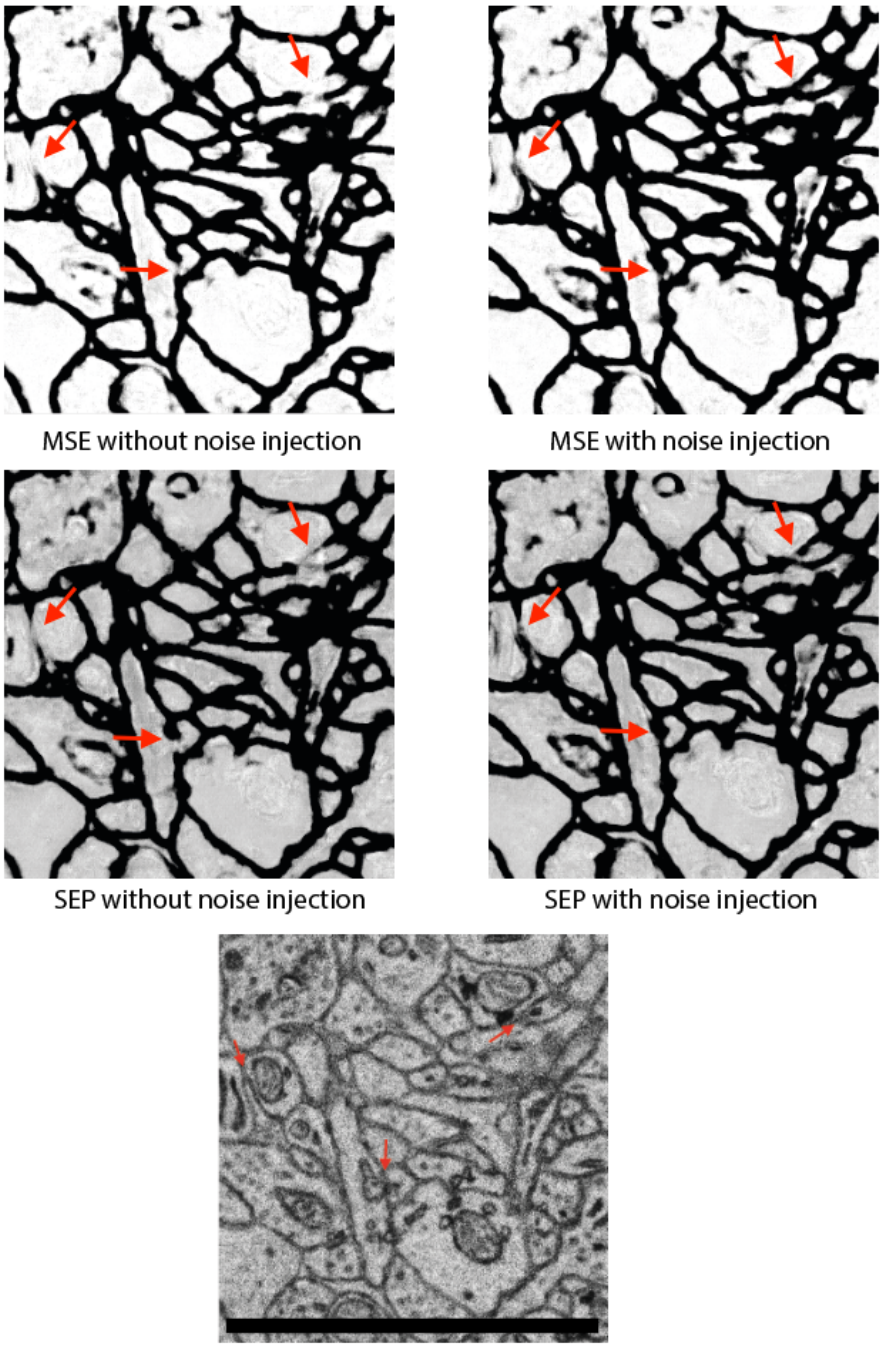


Figure 3.13 – From MSE without noise injection to SEP with noise injection

Addressed MCM errors highlighted with red arrows
Top left: configuration 5
Bottom right: configuration 6
(Table 2.1)
Scale bar: 2.8 µm

Figure 3.14 – Training error on four different configurations (Table 2.1)

Configuration 6 outperformed neighboring and long range affinity networks (configuration 7, 8 and 9)

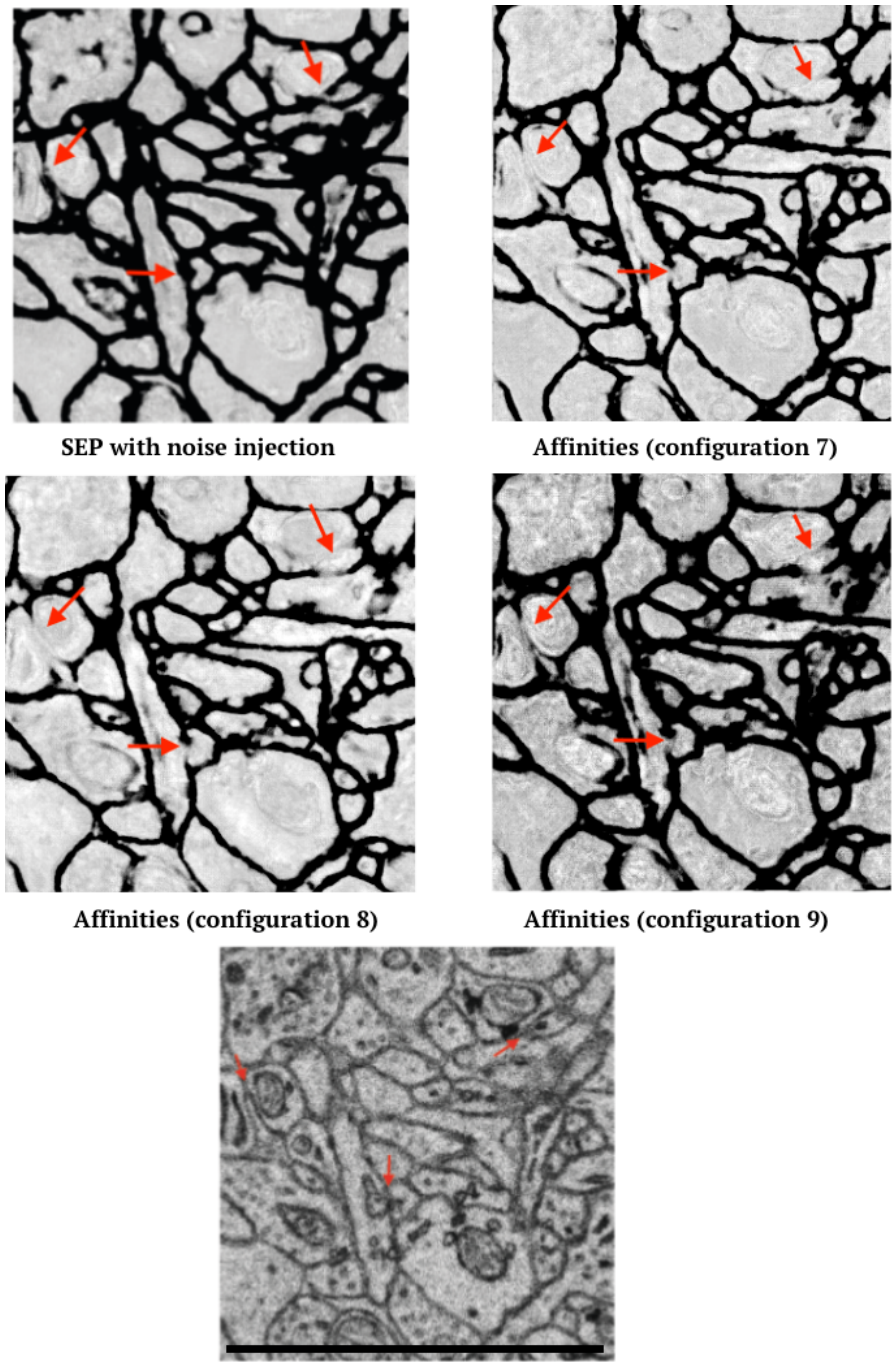
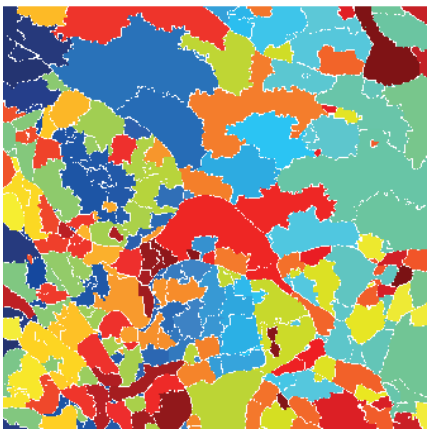
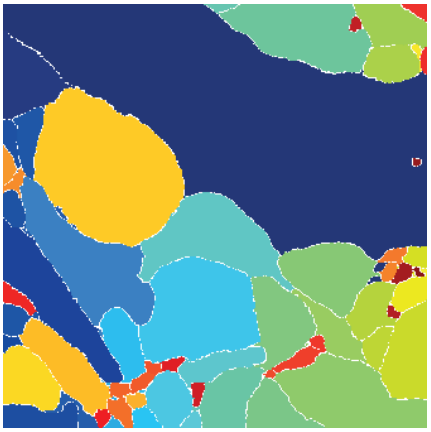


Figure 3.15 – Comparison between configuration 6 (Table 2.1) and short & long range affinity trained networks

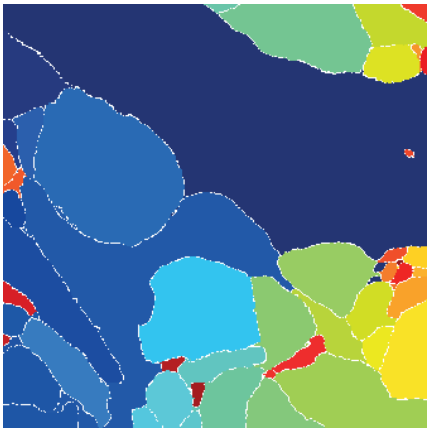
Addressed MCM errors highlighted with red arrows
Decline in probability membrane map accuracy by neighboring and long range affinities.
(Scale bar: 2.8 µm)



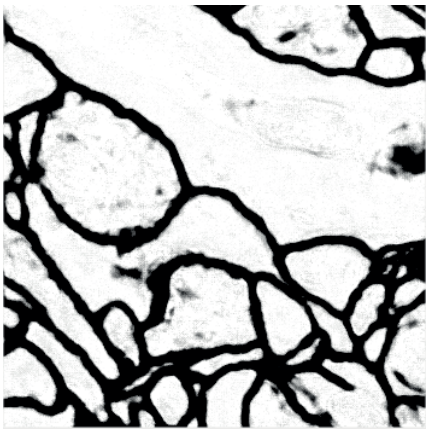
H-minima: 0.11



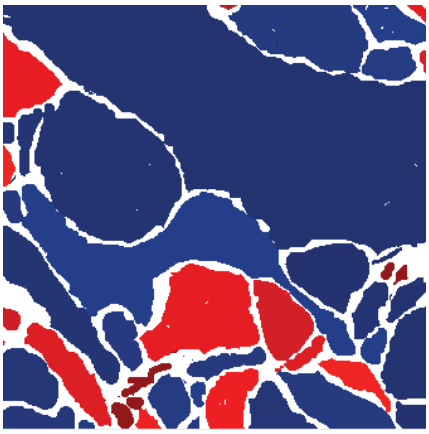
H-minima: 0.25



H-minima: 0.55



CNN predictions



GT volume

Figure 3.16 – Segmentation comparison for different h-minima

CNN used: configuration 5 (Table 2.1) with random cropping
Vmin fixed to 50 voxels
GT volume traced by Namrata Shettar

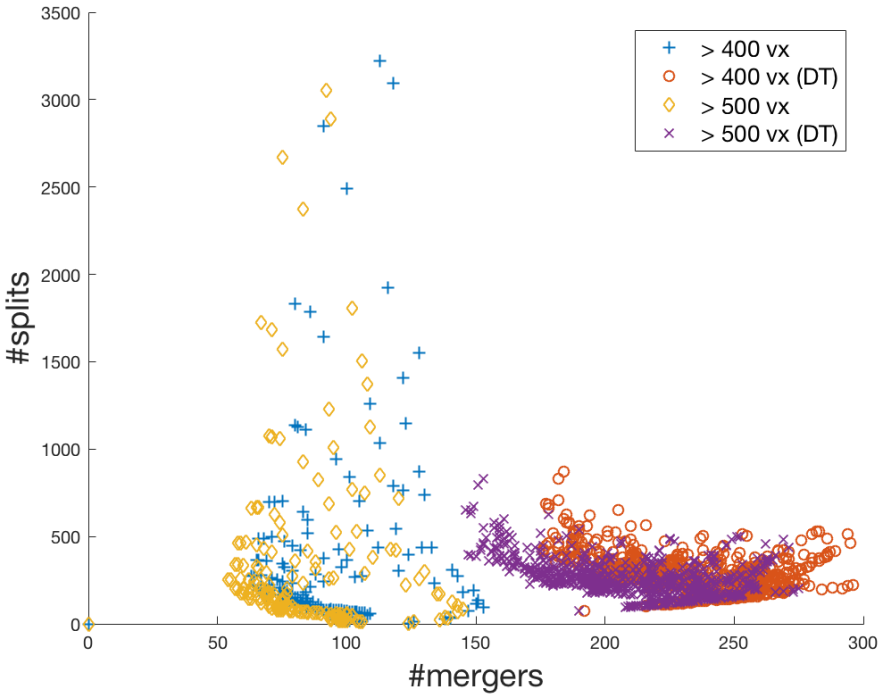


Figure 3.17 – Number of splits and mergers based on volume tracing overlap

The legend corresponds to the different thresholds beyond which the overlap between the ground truth volume tracing and the segmentation is considered as a merger.

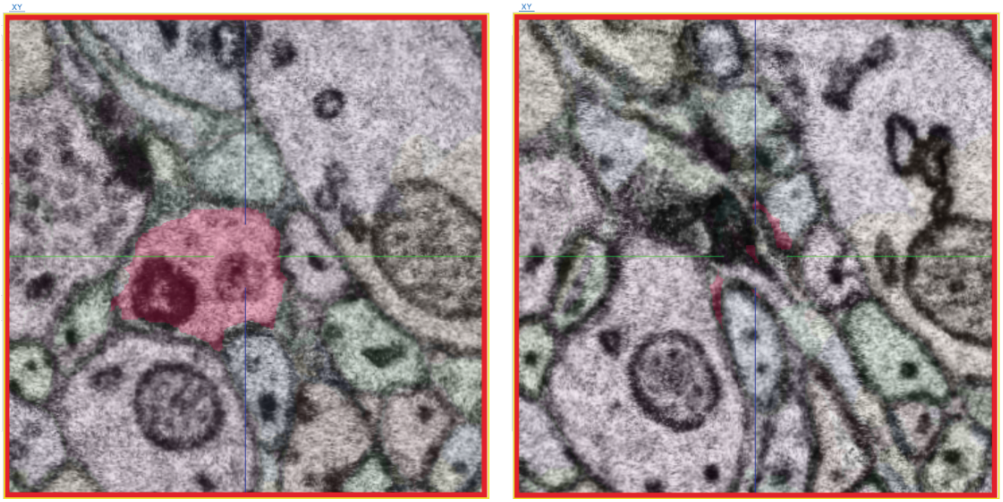


Figure 3.18 – Leaking fragment merger

A small fragment that "leaked" from the highlighted process in red (left) to the subsequent z-neighboring process (right). It is labeled as a merger by the skeleton tracing merge error metric (Berning et al., 2015 [22]).
Box width: 1.5 µm.

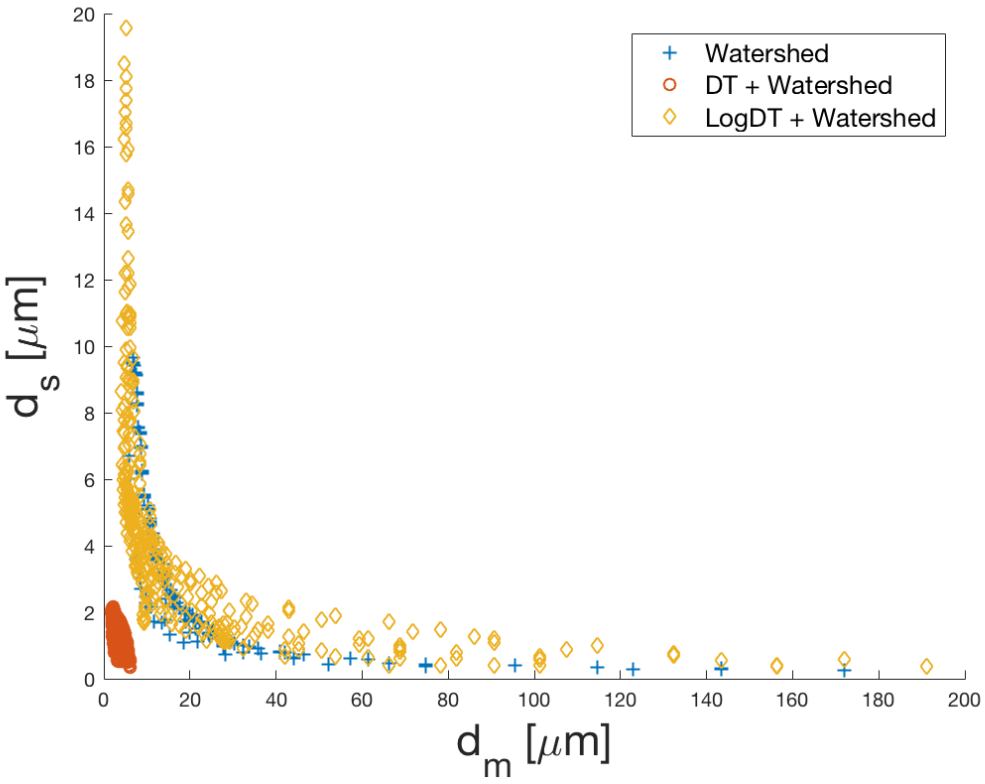


Figure 3.19 – Average distance between splits and mergers based on skeleton tracing

Bounding box size: 5 µm × 5 µm × 4.2 µm
Total path length: 1.721 mm

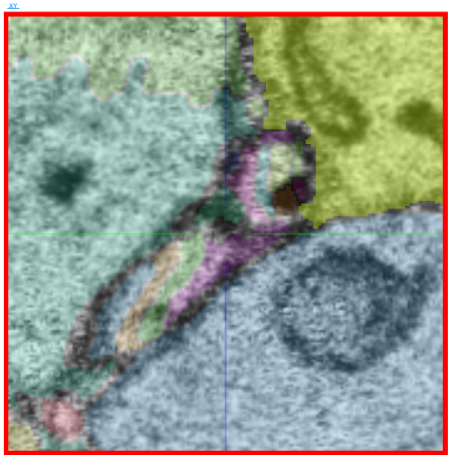


Figure 3.20 – Small splits generated by the log-distance transform watershed

This is a phenotype of a high proportion of split errors, i.e. close to or in membrane and within small processes.
Box width: 700 nm

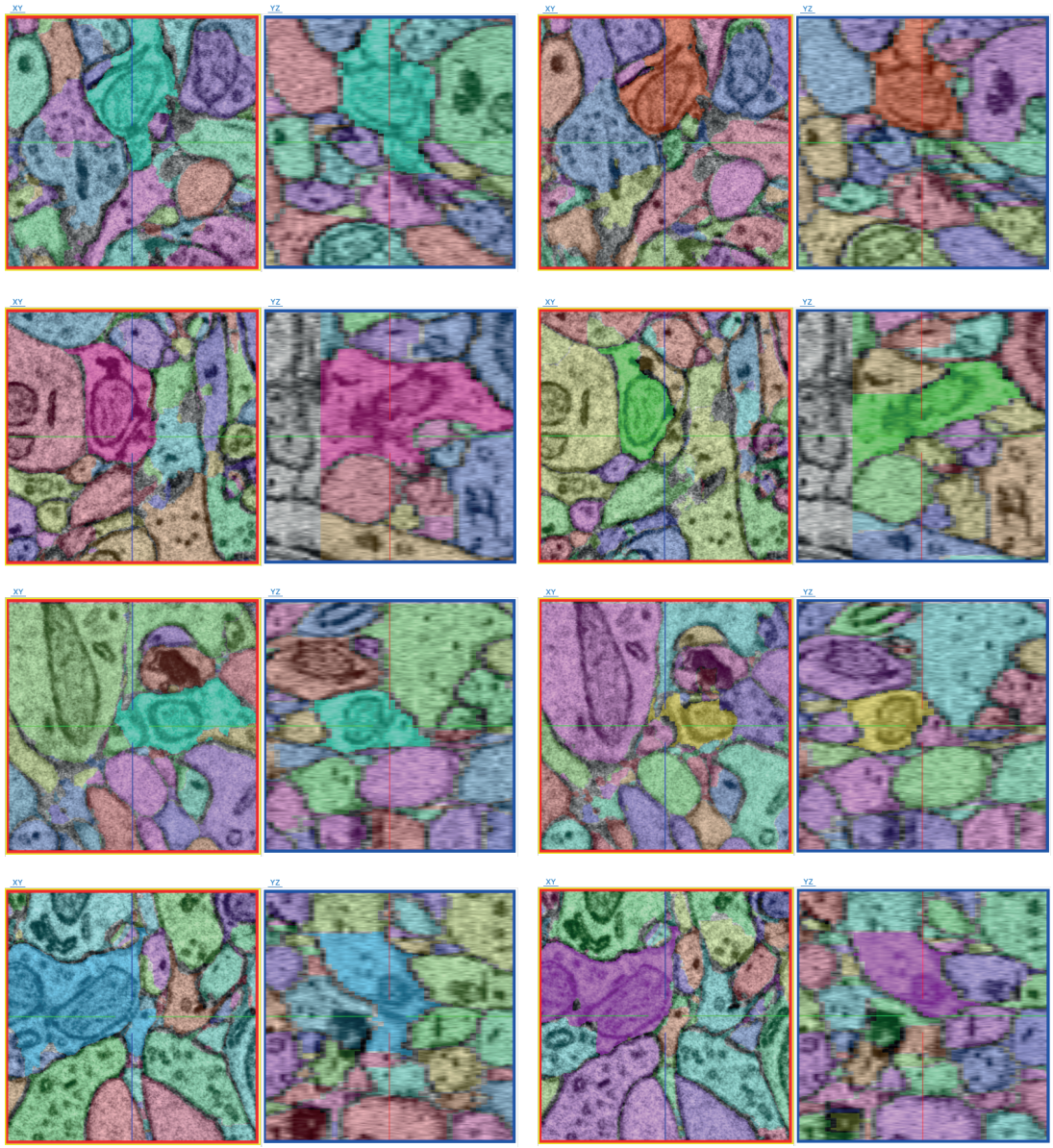


Figure 3.21 – Corrected MCM erros by noise injection and SEP

Left: Segmentation by configuration 5. Right: Segmentation by configuration 6.
Noise injection and SEP efficacy against MCM errors.
Red box: XY axis. Blue box: YZ axis.
Box width: 1.5 µm

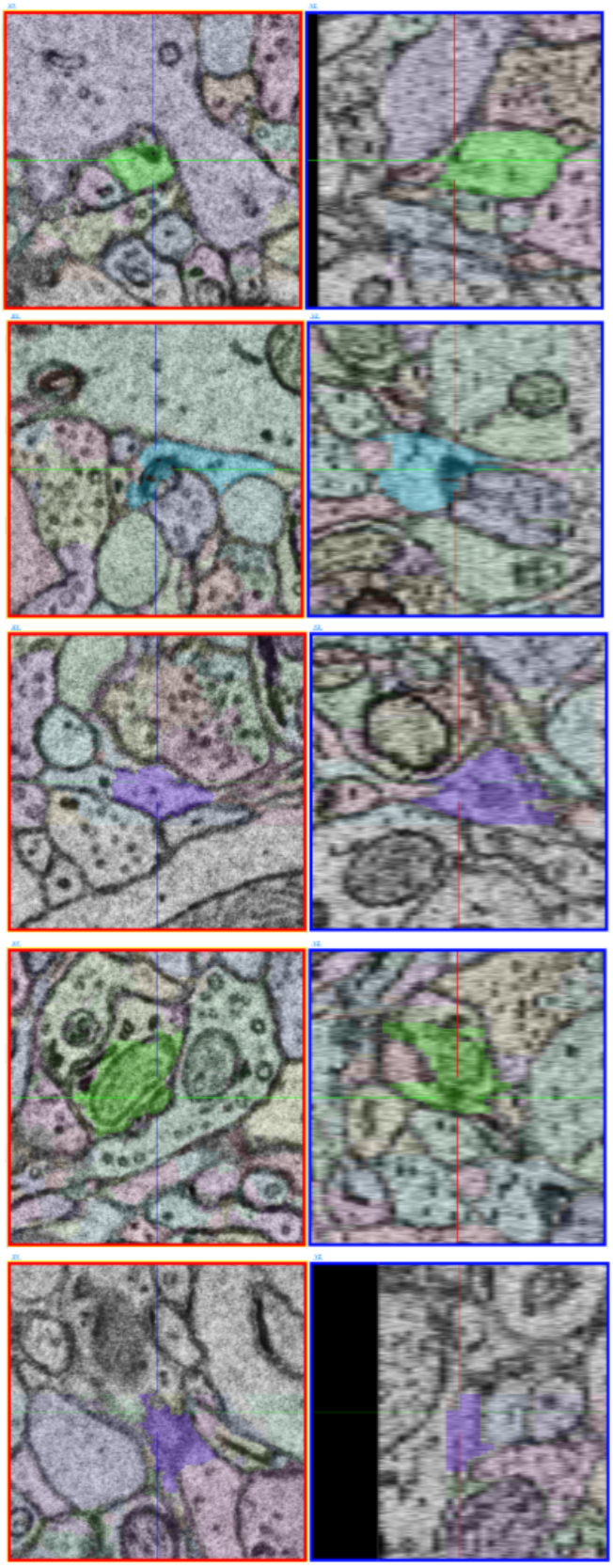


Figure 3.22 – Remaining mergers for best U-Net (configuration 6 in Table 2.1)

LogDT watershed parameters: h-min=0.5 and θ-LogDT=0.7.
Red box: XY axis. Blue box: YZ axis.
Box width: 1.5 µm

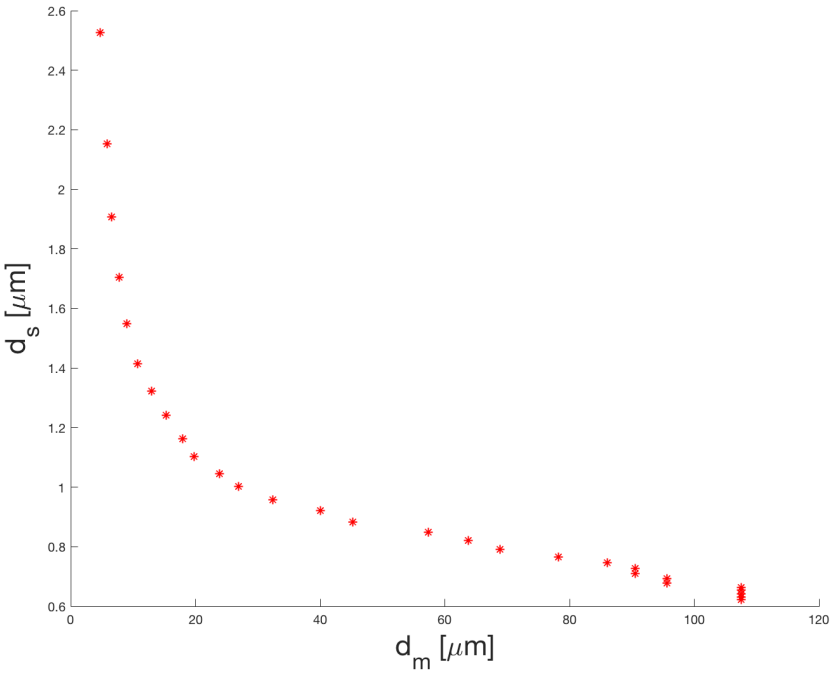


Figure 3.23 – Average distance between splits and mergers for LogDT watershed on different agglomeration thresholds

h-minima: 0.2
θ-LogDT: 0.5

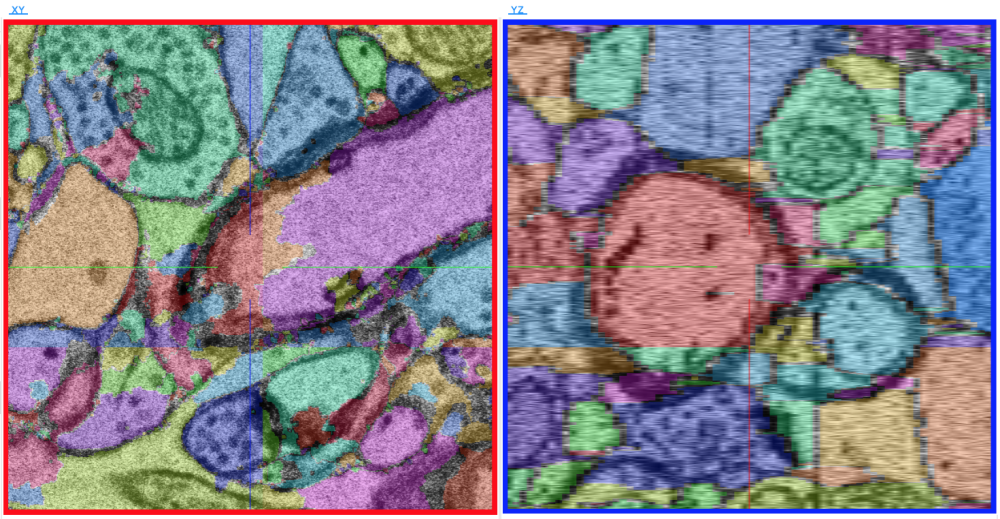


Figure 3.24 – Our segmentation pipeline applied on a new dataset

U-Net in configuration 6 followed by log-distance transform watershed
Red box: XY axis. Blue box YZ axis.
Box width: 2.2 µm

4 Discussion

4.1 Summary

In this work, a variety of CNN paradigms and watershed strategies were tested. First, we have directly applied the lab-based architecture that was designed for SBEM data, SegEM (Berning et al., 2015 [22]). We have then adapted it to mSEM by increasing the FOV of the network. Afterwards, we have decided to switch to the U-Net (Ronneberger et al., 2015 [31]) architecture that is well established for anisotropic EM data. U-Net does not accept arbitrary input shapes which are in fact constrained by its architecture. The latter also defines the FOV of the network that is crucial for its performance. We have varied the number of blocks, the kernel and the max-pooling sizes to come up with a reasonable FOV of 0.7 μm^3 as well as with a number of free parameters that is consistent with the size of our training data to avoid under- or over-fitting problems. To further tackle overfitting, we have applied *dropout* (Srivastava et al., 2014 [39]) to the last layer of the contractive path. Furthermore, we have artificially increased our number of samples by applying different data augmentation techniques: flipping, rotating and pseudo-random shift in brightness and contrast. These two last approaches were replaced by additive and multiplicative gaussian noise, which drastically closed many broken membranes and hence prevented many mergers. In the beginning, we were selecting the training cubes by a sliding window which was then replaced by a random crop. This substitution is motivated by use of all the data spectrum. We have also tried three loss functions, namely mean square error, cross entropy and mean square error with entropy penalty. The latter was implemented to reduce the overconfidence of the network by adding an entropy term to the MSE. Finally, we have enhanced our training labels by applying short and long range affinities with the goal of adding another layer of information beyond voxel classes. This step underperformed solely training our network with the original labels.

The second step in the pipeline is to apply the watershed algorithm in order to map each neurite to a unique segment ID. Directly applying it on the membrane predictions leads to an oversegmentation. Hence we preprocess the CNN maps to only keep local minima which exceed a certain threshold. Since watershed supposes isotropy, we have upsampled the

membrane predictions by spline interpolation, applied watershed, and then use a majority vote to recover the original shape. This approach has proved to be unsuccessful. Solely applying watershed outperformed the prior use of distance transform. However, the log-distance transform followed by a watershed, has proven to be successful by being conservative on regions that are close to membranes. The computation of split and merger rates was not straightforward because of a *bleeding* effect that was observed in the neighboring segments. This is mainly the consequence of the one-voxel boundary that is supposed by the watershed, while our mSEM boundaries are about eight times broader.

The performance was tested on a 5 µm × 5 µm × 4.2 µm bounding box which corresponds to a neurite path length of 1.72 mm. For a split every 0.6 µm, the inter merger distance reached 172 µm. After investigating the detected 10 mergers. We found out that only half are actually true mergers, hence reaching an inter merger distance of 344 µm and a total IED of 600 nm. Merger errors encompass mainly artefact and missing membrane errors. Applying neighboring and long range affinities of up to 105 µm in z did not solve the latter problem. The inspection of the splits shows that they mostly concern small processes rather than large ones. The hierarchical agglomerative step helped reduce the number of splits at the cost of introducing few mergers, even for very conservative thresholds.

4.2 Outlook

In order to generate a connectome, synapse detection using SynEM (Staffler et al., 2017 [28]) is a crucial step that is highly dependant on the segmentation quality. Based on SegEM (Berning et al., 2015 [22]), the fully automated synapse detection tool achieves a 97% precision and recall on neuron-to-neuron connectomes and 88% on single synapse detection. A very high fraction of SynEM errors occured at locations nearby SegEM errors. SynEM errors were reduced by 46% after correcting SegEM one. Hence, our work can drastically contribute to reduce false negatives in synapse detection since it achieved a significantly lower merger rate.

SegEM (Berning et al., 2015 [22]) is also used in the dense connectomic reconstruction of layer 4 of the mouse somatosensory cortex (Motta et al., 2018 [43]). Merge errors were corrected after agglomeration by detecting geometric configurations where two or more neurites are crossing over. It is assumed that these chiasmata configurations occur very rarely in branching neurites. Hence, human annotators were queried to trace from different locations in order to confirm and later solve these potential mergers. In total, this step consumed 1,132 work hours. Our highly merger conservative U-Net based model can thereby drastically reduce this human labour.

Using one *Tesla V100-PCIE-16GB GPU*, it took ~12h to predict boundary maps in a 28 μm^3 cube and less than 3 hours for the segmentation steps of the pipeline (that is originally built for SegEM) in the computer cluster using hundreds of parallelly running CPUs. Given that the multi-beam electron microscope allows the acquisition of volumes in the order of a cubic millimetre in a reasonable amount of time. In order to segment 1 mm^3 with the current

approach, one would need ~8 years for the membrane prediction step using eight similar GPUs and ~15 years for the log-distance transform watershed. Hence, more computer power and more algorithmic optimizations are necessary to scale up. For the latter, one can for instance use the Fourier Transform to perform fast multiplicative operations in the Fourier domain that correspond to convolutions in the spatial domain. Reducing the precision of the floating point operations can further speed up the process.

4.3 Comparison to previous work

Our work is part of an integrative approach to reconstruct neural circuits from Electron Microscopic (EM) images of animal brain. There is a usual tradeoff between EM volumes and the imaging resolution. Small EM volumes can be acquired at high near isotropic resolution by FIB-SEM or SBEM for instance. However, large volumes come at low z-resolution resulting in highly anisotropic images, acquired by ssTEM or ATUM-mSEM.

The problem of reconstructing anisotropic EM brain tissue using CNNs has achieved great improvement recently after the release of the U-Net architecture by Ronneberger et al. [31], who applied it in 2D to the drosophila brain tissue. Their data has a resolution of 4x4x50 nm^3 and hence a large anisotropy factor of 12.5, in opposite to 8.75 for our data. Because of the small size of their training set (6 μm^3), they have applied a variety of augmentation techniques especially elastic deformation that is claimed to be highly important in data scarcity cases. Funke et al. [26] used a 3D U-Net to predict inter-voxel affinities on anisotropic as well as isotropic data (Table 4.1 shows only the anisotropic data). They have trained their network using an extended version of the MALIS loss function (Turaga et al., 2009 [44]). After averaging the affinities for each voxel to generate a membrane prediction map, they applied the distance transform followed by a seeded watershed. Finally, they use a hierarchical agglomeration algorithm similar to ours to reduce the oversegmentation.

Lee et al. [24] used a symmetric version of U-Net where the input and output shape are similar. They applied 3D convolutions in all layers except those where the anisotropy is the highest: the first block of the contractive path and the last block of the expansive path, where they apply 2D convolutions. Similarly to our work, they avoided downsampling in the z-dimension. Additionally to standard augmentation techniques, they have have faked three types of common data generation problems: misalignments, out-of-focus sections and missing sections. The latter can be partially compared to dropout in our work or to noise injection. They have used short and long range affinities of {6, 18, 54, 162} nm for x-y and {29, 58, 87, 116} nm for z which, contrary to us, improved their results. After applying an edge weighted graph watershed (Zlateski & Seung, 2015 [21]), they have used a hierarchical agglomeration algorithm by setting the scores between supervoxels to the mean affinity of all edges between them. To the best of our knowledge, the only published work on mSEM data segmentation today comes from Parag et al. [25], where they have used a similar 3D U-Net architecture to our. The resolution of their data is 3x3x30 nm and had a 2.5 bigger training set.

In order to reduce the anisotropy, they downsampled by a factor of 2 in x-y before feeding the data to the network that was trained by MALIS loss on voxel affinities. The subsequent step uses the same edge weighted graph watershed algorithm (Zlateski & Seung, 2015 [21]) as the one used by Lee et al [24], followed by a delayed agglomerative clustering algorithm (Parag et al., 2015 [45]).

A model called DeepEM3D (Zeng et al., 2017 [23]), that does not make direct use of U-Net (Ronneberger et al., 2015 [31]) but rather a very complex CNN architecture containing, inter-alia, several blocs of inception residual modules. With the motivation of reducing potential merge errors, the authors trained their CNN on enhanced labels. Similarly to our model, they have introduced weights that are inversely proportional to class ratio to tackle the class imbalance problem. The cubes they are working with are of different slices, where depending on this parameter, they either use 2D or 3D convolution. They then combine the CNN output of the different cubes by selecting the maximum voxel. The motivation behind that is again to prevent mergers. They apply a 3D watershed to generate the final segmentation.

Name	Tissue	Imaging	Resolution	Training data (μm^3)	U-Net
Ronneberger et al., 2015 [31]	Drosophila	ssTEM	4x4x50	6	2D
Zeng et al., 2017 [23]	Mouse	SEM	6x6x29	108	-
Funke et al., 2017 [26]	Fly	ssTEM	4x4x40	125	3D
Lee et al., 2017 [24]	Mouse	SEM	6x6x29	108	2/3D
Parag et al. 2018 [25]	Mouse	ATUM-mSEM	3x3x30	108	3D
Our	Mouse	ATUM-mSEM	4x4x35	43	3D

Table 4.1 – Summary of recent anisotropic brain EM data reconstructions

Quantitatively, these approaches (Table 4.1) use the variation of information metric ([24], [25], [26]) or the rand error ([31], [23]) which makes a direct comparison to our split and merger rates not straightforward. Funke et al. [26] applied their network to SBEM data from our department. This dataset contains 279 volumes of 100x100x100 voxels which corresponds to 877 μm^3, i.e. 20 times larger than the training set used for this work. Table 4.2 shows for a fixed inter split distance, how our merger-conservative model, with less training data but high in-plane resolution achieved a higher inter merger error. However, Funke achieved higher average split distances which may suggest that the hierarchical agglomerative algorithm they are using is more efficient. Additionally, as claimed in section 3.3.1, splits in our model concern small processes more than large ones.

Model	Test set	d_s	d_m	IED
Our model on mSEM data	115 μm^3	2.3	44	2.18
SegEM (Berning et al., 2015 [22]) on SBEM data	131 μm^3	2.121	3.951	1.38
Funke et al., 2017 [26] on SBEM data	131 μm^3	2.46	37	2.30

Table 4.2 – Split-merger rate comparison between three models

An orthogonal approach to our model and all those cited above is the Flood-Filling Network

(Januszewski et al., 2016 [29]). It does not use a pipeline that consists of two step, i.e. membrane prediction followed by a segmentation algorithm, but rather abstracts the watershed step directly in an end-to-end paradigm. It uses a single recurrent convolutional neural network to directly map voxel images to unique object masks by iteratively filling a seeded region by all voxels that belong to the same process as the seed. This approach is computationally extremely expensive but has the advantage of making the second step in our pipeline a machine learnable task, as it is the case for the first step. Recently, efforts towards a learnable watershed have been deployed in a non-EM semantic segmentation contexts (Bai et al., 2017 [46]).

4.4 Limitations and future directions

Several undesired phenomena emerged from the fact that the watershed algorithm as well as the distance transform suppose isotropic images. Furthermore, the former supposes a one voxel boundary between the different objects whereas mSEM membranes are about 8 voxels large. The log-distance transform watershed that was used in this work generate many splits in small processes and at the proximity of membranes, even after replicating the slices to artificially establish isotropy. The agglomeration yields better split rates yet at the cost of introducing new mergers even at conservative thresholds. Using anisotropic versions of watershed and distance transform can potentially reduce the number of splits and avoid the bleeding effect (Section 3.3.1) that hampers direct use of split and merger rates (Berning et al., 2015 [22]).

An approach we did not investigate in this work that might be interesting to test concerns the segmentation strategy (Zlateski & Seung, 2015 [21]) that was used in Lee et al., 2017 [24] and Parag et al. 2018 [25] papers where an edge-weighted graph implementation of watershed algorithm has been used to generate an initial oversegmentation, followed by an agglomerative clustering algorithm.

The majority of our merge errors are due to data artefacts which can be further minimised by extensive data augmentation. For example, elastic deformation is claimed by Ronneberger et al. [31] to play a drastic role in case of training data scarcity. Emulating these data problems by for instance artificially reproducing the class of artefact they are representing can also be beneficial for the networks to be trained on (Lee et al., 2017 [24]). All these techniques can further increase the robustness of the network and its generalization power to different datasets. The second type of merge errors produced by our model concern missing membrane slices, thereby our membrane classification method to solve these errors can be by essence obsolete for this particular problem. Using affinity predictions can potentially be efficient (Parag et al. 2018 [25]) on mSEM data reconstruction.

Bibliography

[1] Hemberger, M., Pammer, L., and Laurent, G. (2016). Comparative approaches to cortical microcircuits. *Current Opinion in Neurobiology,* 41:24-30

[2] Sporns, O., Tononi, G., and Ktter, R. (2005). The human connectome: A structural description of the human brain. *PLoS computational biology,* 1:e42.

[3] Cajal, S. R. (1888). *Estructura de los centros nerviosos de las aves.* Jimnez y Molina.

[4] Golgi, C. (1873). *Sulla struttura della sostanza grigia del cervello. Gazzetta Medica Italiana. Lombardia.* 33:244-246.

[5] Gray, E. G. (1959). Axo-somatic and axo-dendritic synapses of the cerebral cortex an electron microscope study. *Journal of Anatomy,* 93, 420-433.

[6] White, J. G., Southgate, E., Thomson, J. N., and Brenner, S. (1986). The structure of the nervous system of the nematode caenorhabditis elegans. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences,* 314:1-340.

[7] Varshney, L. R., Chen, B. L., Paniagua, E., Hall, D. H., and Chklovskii, D. B. (2011). Structural properties of the caenorhabditis elegans neuronal network. *PLoS computational biology,* 7(2):e1001066.

[8] Ward, S., Thomson, N., White, J. G., and Brenner, S. (1975). Electron microscopical reconstruction of the anterior sensory anatomy of the nematode caenorhabditis elegans. *The Journal of comparative neurology,* 160:313-337.

[9] Denk, W. and Horstmann, H. (2004). Serial block-face scanning electron microscopy to reconstruct three-dimensional tissue nanostructure. *PLoS biology,* 2:e329.

[10] Briggman, K. L., Helmstaedter, M., and Denk, W. (2011). Wiring specificity in the direction-selectivity circuit of the retina. *Nature,* 471:183-188.

[11] Helmstaedter, M., Briggman, K. L., Turaga, S. C., Jain, V., Seung, H. S., and Denk, W. (2013). Connectomic reconstruction of the inner plexiform layer in the mouse retina. *Nature,* 500:168-174.

[12] Takemura, S.-y., Bharioke, A., Lu, Z., Nern, A., Vitaladevuni, S., Rivlin, P. K., Katz, W. T., Olbris, D. J., Plaza, S. M., Winston, P., Zhao, T., Horne, J. A., Fetter, R. D., Takemura, S., Blazek, K., Chang, L.-A., Ogundeyi, O., Saunders, M. A., Shapiro, V., Sigmund, C., Rubin, G. M., Scheffer, L. K., Meinertzhagen, I. A., and Chklovskii, D. B. (2013). A visual motion detection circuit suggested by drosophila connectomics. *Nature,* 500:175-181.

[13] Schmidt, H., Gour, A., Straehle, J., Boergens, K. M., Brecht, M., and Helmstaedter, M. (2017). Axonal synapse sorting in medial entorhinal cortex. *Nature,* 549(7673):6-9.

[14] Hayworth, K., Kasthuri, N., Schalek, R., and Lichtman, J. (2006). Automating the collection of ultrathin serial sections for large volume tem reconstructions *Microscopy and Microanalysis,* 12(S02):86.

[15] Helmstaedter (2013). Cellular-resolution connectomics: challenges of dense neural circuit reconstruction. *Nature Methods,* 10, 501-507.

[16] Eberle, A., Mikula, S., Schalek, R., Lichtman, J., Tate, M. K., and Zeidler, D. (2015). High-resolution, high-throughput imaging with a multibeam scanning electron microscope. *Journal of microscopy,* 259(2):114-120.

[17] MultiSEM 505/506 The World?s Fastest Scanning Electron Microscopes. *URL:https://www.zeiss.com/microscopy/int/products/scanning-electron-microscopes/multisem.html*

[18] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation,* 1(4):541-551.

[19] Turaga, S., Briggman, K., Denk, W., Seung, S., and Helmstaedter, M. (2009). Maximin affinity learning of image segmentation. *In Advances in Neural Information Processing Systems,* pages 1865-1873.

[20] Jain, V., Bollmann, B., Richardson, M., Berger, D. R., Helmstaedter, M. N., Briggman, K. L., Denk, W., Bowden, J. B., Mendenhall, J. M., Abraham, W. C., et al. (2010). Boundary learning by optimization with topological constraints. *In Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference.* on, pages 2488-2495. IEEE.

[21] Zlateski. A and Seung. H. S, (2015). Image segmentation by size-dependent single linkage clustering of a watershed basin graph. *arXiv:* 1505.00249.

[22] Berning, M., Boergens, K. M., and Helmstaedter, M. (2015). SegEM: efficient image analysis for high-resolution connectomics. *Neuron,* 87(6):1193-1206.

[23] Zeng, T., Wu, B., and Ji, S., (2017). DeepEM3D: Approaching human-level performance on 3D anisotropic EM image segmentation. *Bioinformatics,* 15;33(16):2555-2562.

[24] Lee, K., Zung, J., Li, P., Jain, V., Seung. H. S. (2017). Superhuman Accuracy on the SNEMI3D Connectomics Challenge *arXiv:* 1706.00120.

[25] Parag, T., Tschopp, F., Grisaitis, W., Turaga, S., Zhang, X., Matejek, B., Kamentsky, L,., Lichtman, J., Pfiste, H. (2018). Anisotropic EM Segmentation by 3D Affinity Learning and Agglomeration. *arXiv:* 1707.08935.

[26] Funke, J., Tschopp, F. D., Grisaitis, W., Singh, C., Saalfeld, S., and Turaga, S. C. (2017). A deep structured learning approach towards automating connectome reconstruction from 3d electron micrographs. *arXiv:* 1709.02974.

[27] Boergens, K. M., Berning, M., Bocklisch, T., Braunlein, D., Drawitsch, F., Frohnhofen, J., Herold, T., Otto, P., Rzepka, N., Werkmeister, T., Werner, D., Wiese, G., Wissler, H., and Helmstaedter, M., (2017). webknossos: efficient online 3d data annotation for connectomics. *Nature Methods,* 14:691-694.

[28] Staffler, B., Berning, M., Boergens, K. M., Gour, A., van der Smagt, P., and Helmstaedter, M. (2017). SynEM: Automated synapse detection for connectomics. *eLife,* 6:e26414.

[29] Januszewski, M., Maitin-Shepard, J., Li, P., Kornfeld, J., Denk, W., and Jain, V. (2016). Flood-Filling Networks. *arXiv preprint arXiv:* :1611.00421.

[30] Meyer, F. (1994). Topographic distance and watershed lines. *Signal Processing,* 38(1):113-125.

[31] Ronneberger, O., Fischer, P., and Brox, T., (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *In Medical Image Computing and Computer-Assisted Intervention (MICCAI),* volume 9351 of LNCS, pages 234-241.

[32] Kubota, Y., Sohn, J., Hatada, S., Schurr, M., Straehle, J., Gour, A., Neujahr, R., Miki, Mikula, S. and Kawaguchi, Y. (2018). A carbon nanotube tape for serial-section electron microscopy of brain ultrastructure. *Nature Communications,* 9:1-3.

[33] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review,* 65(6):386-408.

[34] Hochreiter, S., and Schmidhuber, J., (1997). Long short-term memory. *Neural computation 9,* (8): 1735–1780.

[35] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X., and Brain, G. (2016). TensorFlow: A System for Large-Scale Machine Learning. *In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI?16),* pages 265-284.

[36] Glorot, X., Bengio, Y., (2009). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, PMLR,* 9:249-256.

[37] Vapnik, V. (1998). Statistical Learning Theory. Adaptive and learning systems for signal processing, communications, and control.

[38] Bartlett, P. and Mendelson, S. (2003). Rademacher and gaussian complexities: risk bounds and structural results. *Journal of Machine Learning Research,* 3:463-482.

[39] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research,* 15(1):1929-1958.

[40] Classic Watershed - ImageJ
URL: https://imagej.net/Classic_Watershed

[41] Hinton, G., Srivastava, N., and Swersky, K. (2012). Neural Networks for Machine Learning, lecture 6a. Overview of mini-batch gradient descent.
URL: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[42] Sutskever, I., Martens, J., Dahl, G., Hinton G., 2013. On the importance of initialization and momentum in deep learning. *In Proceedings of the 30th International Conference on Machine Learning (ICML-13)* pp. 1139-1147.

[43] Motta, A., Berning, M., Boergens, K. M., Staffler, B., Beining, M., Loomba, S., Schramm, C., Hennig, P., Wissler, H. & Helmstaedter,.M. (2018). Dense connectomic reconstruction in layer 4 of the somatosensory cortex (under review).

[44] Turaga, S. C., Briggman., K. L., Helmstaedter, M., Denk, W., & Seung H. S. (2009). Maximin affinity learning of image segmentation *Neural Information Processing Systems (NIPS).*

[45] Parag, T., Chakraborty, A., Plaza, S., and Scheffer, L. (2015). A context-aware delayed agglomeration framework for electron microscopy segmentation. *PLoS ONE:* 10(5).

[46] Bai, M., and Urtasun, R. (2017). Deep watershed transform for instance segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* pp. 2858-2866.