ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Learning to Agglomerate in Region Adjacency Graphs for Electron Microscopy Neuron Segmentation using Graph Neural Networks

Master's Thesis

Benjamin Gallusser

September 15, 2019

Advisors: Prof. Dr. Angelika Steger, Dr. Jan Funke, Dr. Matthew Cook
Department of Computer Science, ETH Zürich

Abstract

Fully automating neural circuit reconstruction from electron microscopy (EM) data has the potential to provide a huge leap for neuroscience. The final step of many neuron reconstruction pipelines consists of agglomerating small compact fragments to form entire neurons. This work explores using Graph Neural Networks to learn that task from the Region Adjacency Graph (RAG) of said fragments, combined with features learned by a Convolutional Neural Network directly from EM data. For this purpose we present a generalizable, geometric Graph Neural Network called *RagNet*. It labels RAG edges as *merge* or *split*, which allows the direct extraction of segmentations. In contrast to previous agglomeration methods, the *RagNet* allows an increase in the context considered for making an edge merge decision. This benefit is empirically confirmed by applying *RagNet* on top of an edge-wise prediction method and increasing the class-balanced accuracy by ten percentage points. Lastly, we fuse our implementation with an existing large-scale neuron reconstruction pipeline and report initial results on a recently imaged *Drosophila melanogaster* brain dataset.

Acknowledgements

First of all, I am grateful to my direct advisor Jan Funke for his excellent mentorship, for the remarkably quick and spot-on help when I was facing difficulties and in general for the fruitful discussions we had throughout this work. Second, I would like to thank Angelika Steger, Matthew Cook and Julia Buhmann for enabling and supporting this collaboration.

Next, I wish to thank Arlo Sheridan for supporting me in integrating my work with the Funke Lab neuron reconstruction pipeline, for boosting the aesthetics of my figures and especially for helping me to understand Connectomics and Neuroscience. I very much appreciated the friendly and driving atmosphere in the Funke Lab at HHMI Janelia, all its members have made this thesis some exciting and fun months of research for me. Equally, I would like to thank the members of Matthew Cook's Cortical Computation group at the INI for their enjoyable and inspiring company when I was working in Zurich.

Furthermore, I would like to acknowledge the FlyEM team at HHMI Janelia for letting us work on their newest dataset. Moreover, I am grateful to HHMI Janelia's Visiting Scientist Program for covering some of my expenses related to working and living on the Janelia Research Campus. After many months of intense use of it, I also wish to acknowledge Matthias Fey for the creation and extraordinarily attentive maintenance of *PyTorch Geometric*, which is the base of our implementation. Finally, I would like to thank Tess Oram and Nils Eckstein for providing feedback on the draft version of this thesis.

Contents

Chapter 1

Introduction

1.1 Connectomics

How does the brain work in its innermost elements?

Early advances to answer this question were made in the late 19th century by neuroscientists such as Ramón y Cajal and Golgi [7]. Since then, generations of scientists have continued to improve the common understanding of the brain at micro- and macroscopic scale. One area of ongoing research is the detailed structural analysis of brains from various organisms, as a dominant hypothesis is that the connectivity of neurons is crucial to computation, learning and memory. To obtain a brain's neuronal wiring diagram, also called *Connectome* [38], it is necessary to reconstruct each neuron and to detect the exact location of all synapses. This is possible thanks to Electron Microscopy (EM), which allows us to image a stack of 2D images from fixed brain tissue at nanometer resolution (example in Figure 1.1). The image stack is afterwards virtually assembled to form a 3D reconstruction.

The reconstruction of the roughly 300 neurons and 7500 synapses forming the complete nervous system[1] of the *C. elegans*[2] took years of human labor [42], which shows the need for automated reconstruction methods for studying larger organisms. Additionally, electron microscopy has experienced major speedups over the past decades, displayed by the recent acquisition of a full adult *Drosophila melanogaster*[3] brain, containing roughly 10^5 neurons, over the period of just 16 months [44]. This dataset, also known as FAFB[4],

[1] The nervous system includes the brain, the spinal cord (if present) and the nerves throughout the body of an animal.

[2] *C. elegans* is a transparent worm of approximately 1 mm in length and an extensively studied model organism.

[3] *Drosophila melanogaster*, commonly known as fruit fly, is a model organism widely used in biological research measuring in at ca. 2.5 mm in length.

[4] FAFB is the acronym for (female) *Full Adult Fly Brain*.

consists of roughly 21 million individual images amounting to 106 terabytes of data.

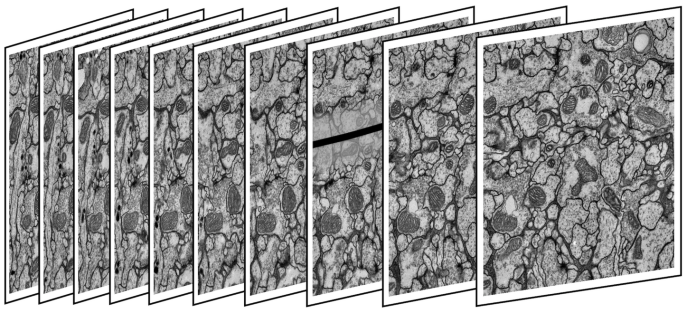


Figure 1.1: Contiguous stack of electron microscopy data, from [11]

Supported by a multitude of automated methods that we introduce in the following sections, we are now able to extract an imperfect segmentation for a dataset of this size. To obtain a full, error-free connectome of a *Drosophila* brain, hundreds of years of accumulated human effort are still required to proof-read and correct a current automated segmentation. Therefore, reconstructing the connectome from larger model organisms such as a mouse ($\sim 10^7$ neurons) or a zebra finch ($\sim 10^8$ neurons), let alone from a human ($\sim 10^{10}$ neurons), remains out of reach. However, all aspects of Connectomics are rapidly advancing and the field will keep moving the boundaries of what is possible in neuroscience.

1.2 Data Acquisition

1.2.1 Electron microscopy

Electron microscopy currently provides the best imaging quality at nanometer resolution [22]. Currently, there are two main methods of high-throughput EM for imaging brain tissue in Connectomics, serial-section transmission electron microscopy (SSTEM) and focused ion beam scanning block-face electron microscopy (FIB-SBEM).

For both methods the brain tissue is initially removed from the organism, chemically fixed to preserve it and chemically stained to improve imaging contrast.

For the first approach, SSTEM, a block of brain tissue gets sliced into single sections with a thickness of merely tens of nanometers. This is a delicate, error-prone procedure and also leads to anisotropic resolution, as the resolu-

tion in x- and y-direction[5] is usually in single-digit nanometers. For example, the full adult *Drosophila melanogaster* dataset mentioned in Section 1.1 was imaged at $4 \times 4 \times 40$ nm resolution. This can be problematic for automated reconstruction, because some objects of interest such as the cell membrane of neurons might be thinner than 40 nm.

The alternative, FIB-SBEM, does not require sliced sections, but instead images are acquired directly from the face of the tissue block, of which we can periodically remove a layer of single-digit nanometer thickness with a focused ion beam (FIB). This technique is predestined to acquire isotropic datasets, such as the $8 \times 8 \times 8$ nm dataset used for this work, which we will introduce in more detail in Section 1.4. The major present downside of FIB-SBEM is the increased cost compared to SSTEM. Moreover, the field-of-view or FIB-SBEM, meaning the size of the block surface, is limited. This requires cutting the *Drosophila* brain multiple times in z-direction with the so-called *Hot Knife* method [15], which will destroy a small amount of tissue at each cut surface. Overall, the substantial reduction of imaging artifacts and the data isotropy make this the current state-of-the-art imaging method.

1.2.2 Alignment of image stack

The first computational step is assembling all 2D images to form a 3D volume. As the field of view of the microscope might be smaller than an entire 2D section we want to image, we actually get multiple overlapping images per section and have to stitch them together. This task is e.g. addressed in [19] and yields nearly perfect results.

Next, we have to align all sections in z-direction, which is usually harder than the stitching in 2D from above. Common problems are changes in translation and rotation, non-linear deformations of the sections and also incorrect order of the sections. This step has turned out to be crucial for current neuron segmentation approaches [18]. A state-of-the-art approach is detailed in [20].

1.3 Automated Neuron Segmentation

The Computer Vision community is working on multiple problems that involve localizing objects in images. *Object Detection* denotes finding objects of a specific type, e.g. cars, in an image and drawing a rectangular box around each of them. In *Semantic Segmentation*, boxes are not sufficient, but instead each pixel in an image is labeled with one of multiple predefined classes, e.g. as car, person or road. When different objects of the same type

[5] The two visible dimensions in a single image in EM are commonly referred to as x- and y-direction, whereas the thickness of each imaged section is called z-direction.

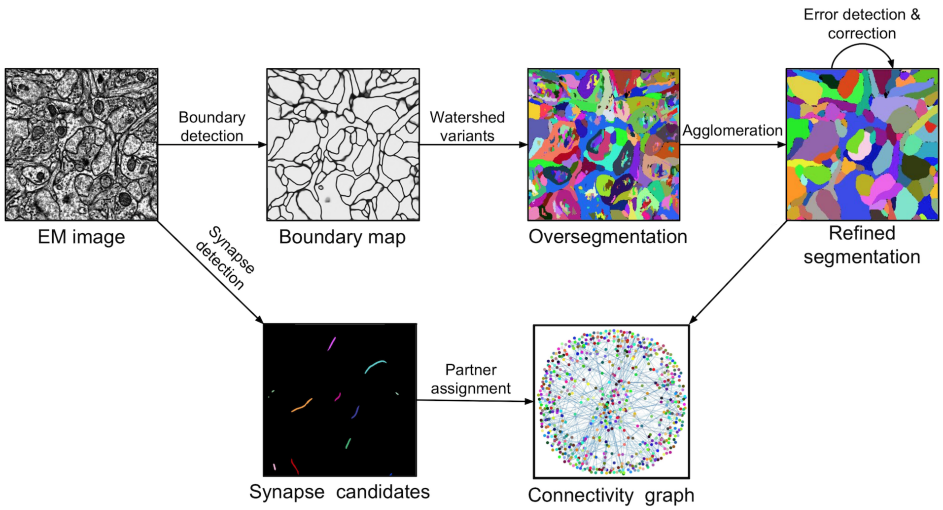


Figure 1.2: Typical EM image processing pipeline in Connectomics, adapted from [23]

have to be clearly separated in an image, e.g. two cars next to each other, we call this task *Instance Segmentation*. Automated neuron segmentation is an Instance Segmentation problem with two critical adaptions. Classic Instance Segmentation is performed on 2D images and compact objects [25], whereas automated neuron segmentation operates in 3D and has to segment very delicate and branched-out objects. This calls for methods different from common Instance Segmentation approaches in the literature.

Most automated neuron segmentation pipelines are divided into three major steps: Boundary detection, oversegmentation[6] into small fragments and agglomeration of these fragments to form complete segments. This pipeline is illustrated in Figure 1.2.

A different, usually independent task for obtaining a connectome is synaptic partner prediction. The locations of pre- and post-synaptic partners are detected with Machine Learning algorithms such as Convolutional Neural Networks [5] and in turn mapped onto segmented neurons to obtain a complete connectome (refer to Figure 1.2). We will not discuss this in detail in this work.

The first pipeline step is neuronal boundary detection in the aligned 3D image stack, usually by means of supervised machine learning. Some of the leading approaches in the field [12, 1, 24] use a Convolutional Neural

[6] An oversegmentation of an image implies that objects can be split into multiple fragments. Conversely, an undersegmentation would exhibit fragments that contain parts from more than one object, making a perfect reconstruction impossible.

Networks (CNN) for this task, most commonly a variant of the U-net architecture [33]. Instead of predicting for each voxel whether it is part of a neuronal boundary or not, it has shown to be beneficial to predict values for edges in an *affinity graph* as introduced by [39]. The affinity graph connects voxels directly next to each other and exhibits a boundary probability value for each edge. As an extension, the authors of [24] sparsely connect voxels in some extended local neighborhood.

If the resulting boundaries were perfectly accurate, it would be enough to extract segments that are not interrupted by any boundary to obtain a perfect segmentation. However, detecting neuronal boundaries is prone to multiple types of errors. First, the EM images also contain boundaries of subcellular structures such as mitochondria. These boundaries can lead to incorrectly splitting neurons. Second, the raw data might be imperfect due to artifacts, problems with staining or insufficient resolution for thin parts of membranes.

Note that only a few local errors for boundary prediction can completely falsify the global segmentation. Therefore, common approaches perform additional steps after boundary prediction that make use of an extended context. In order to maintain the possibility to obtain a perfect segmentation, boundary prediction is rather aggressive, meaning that in case of doubt a boundary is predicted. It's much easier to remove false boundaries compared to adding a missing boundary in the subsequent steps.

The second step is to use the boundary predictions to partition the image stack into small fragments. This is performed with watershed-type algorithms, a good overview can be found in [23]. The idea in a watershed algorithm is to start growing fragments from some initial set of points and then iteratively increase them to contain neighboring voxels if their boundary probability is below a certain threshold. Again, there is a tradeoff between fragment size and upper bound segmentation accuracy in this step.

The third and final step consists of agglomerating the fragments to form the final segmentation. For this, the fragments are represented as a region adjacency graph (RAG), where each fragment is a vertex and two touching fragments are represented as an edge. The RAG enables the use of deterministic graph-based algorithms for agglomeration. Alternatively, there are learned approaches that also consider features not contained in the RAG. A short overview of approaches is available in [23]. Furthermore, we describe some approaches in more detail in Section 2.1.

A fundamentally different approach to neuron segmentation is presented as *Flood-filling network* (FFN) in [18]. Instead of extracting all segments present in a fixed-sized window, each object is segmented individually by recursively expanding a binary mask using a CNN. The local CNN input window

is moved over the image stack until an object is completely recovered, making further processing required by boundary detection obsolete. While eliminating tricky task-dependencies inherent to boundary prediction pipelines seems to be beneficial, note that there is no direct comparison to competing state-of-the-art methods in the literature that confirms the claimed superiority of FFN by an order of magnitude[7]. Furthermore, FFN is computationally far more expensive than boundary prediction approaches, as the number of times each voxel is processed is not bounded.

1.4 The Hemibrain dataset

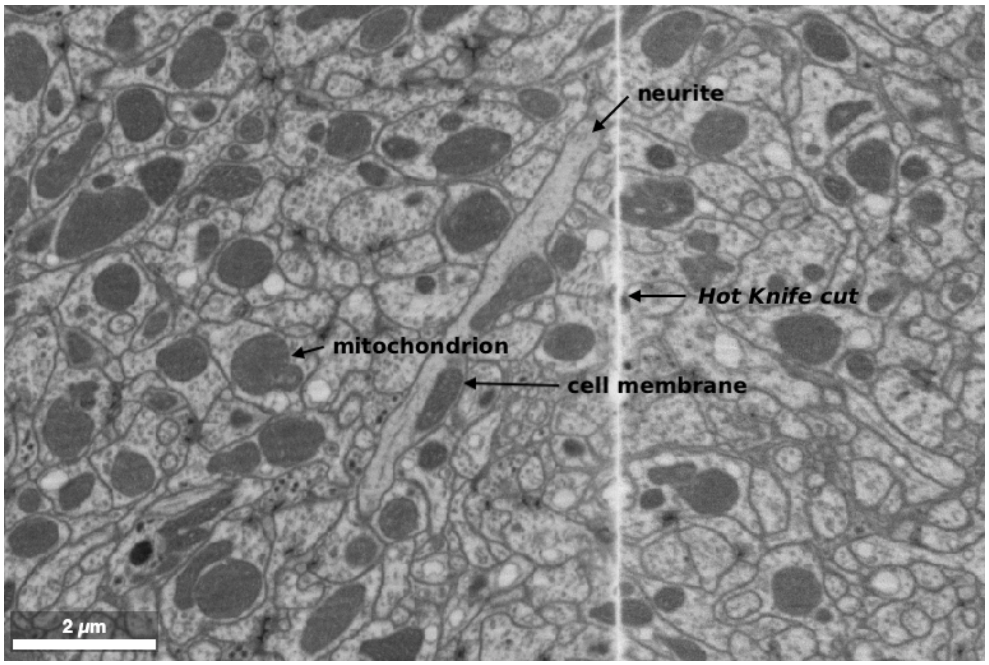


Figure 1.3: Example electron microscopy data, by courtesy of the FlyEM project at HHMI Janelia.

The dataset that we use in this work is a FIB-SBEM dataset of roughly a third of the *Drosophila melanogaster* brain volume, named *Hemibrain* [10]. It has been imaged at $8 \times 8 \times 8$ nm resolution by the FlyEM team[8] at HHMI Janelia and is the largest FIB-SBEM brain dataset as of now. The motivation for recording it is the hypothesis that the anisotropic resolution and the large number of artifacts in SSTEM datasets pose a challenge that is too

[7] https://ai.googleblog.com/2018/07/improving-connectomics-by-order-of.html
[8] https://www.janelia.org/project-team/flyem

hard for current automated reconstruction methods. Thus, the easiest way to improve state-of-the-art reconstructions is to improve the data quality.

The total volume of *Hemibrain* is roughly $2.5 \times 10^7\,\mu m^3$, which amounts to approximately 50 terabytes of image data containing over 20,000 neurons[9].

The FlyEM team is currently working on proof-reading a segmentation created by the authors of [18]. We can use the neurons that have already been proof-read as ground truth data for our supervised machine learning algorithms.

Specifically, we pick three cubes from the ellipsoid body[10] with respective volume sizes of $\sim$1600 μm^3, $\sim$4700 μm^3 and $\sim$10,400 μm^3 for training and evaluating our methods.

1.5 Affinity-based neuron segmentation pipeline

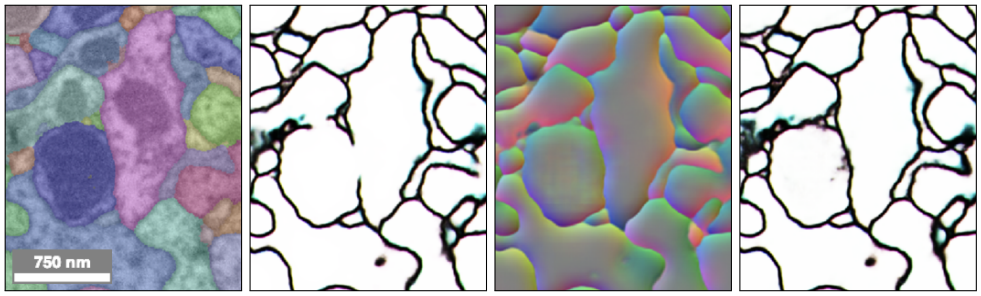
The work presented in this thesis is built on top of the neuron segmentation approach presented in [12], which we will briefly summarize in this section. The method follows the common three-step approach based on boundary predictions from Section 1.3. The neuron boundaries are created by predicting affinities in the range from 0 to 1 between neighboring voxels with a 3D U-Net [6]. The loss function of the U-Net is a structured loss based on MALIS [3], which describes the notion of a topological error on the affinity graph using connected component analysis. Next, the approach uses a standard seeded watershed algorithm to extract the fragments that form the initial oversegmentation. The corresponding Region Adjacency Graph is built with one minus the maximum affinity between neighboring fragments as edge scores. Thereafter, a two-step agglomeration approach is employed. First, all edges with a score lower than a threshold $\varepsilon = 0.1$ are merged, before a hierarchical agglomeration scheme is employed. This means that whenever multiple edges have to be merged as a result of merging two fragments, the score of all involved edges is updated with a quantile value of these scores, e.g. the median. Ultimately, a thresholded connected component algorithm on the updated edge scores is used to create the final segmentation.

The method for boundary prediction has been improved recently by predicting *Local shape descriptors* (LSD) [36] as an auxiliary learning task in order to aid the established affinity prediction with a U-Net. The LSDs are low-dimensional vectors that capture basic features of each voxel relative to the

[9] https://www.janelia.org/project-team/flyem/blog/my-em-data-is-segmented-now-what

[10] The ellipsoid body in the *Drosophila* brain plays a crucial role for sensory integration and motor coordination.

neuron it is part of. The target LSDs are pre-calculated from volumetric ground truth data. Compared to the sharp color gradients of affinities, they provide smoother targets, which makes boundary predictions more robust (refer to Figure 1.4). Therefore, we use the RAG created with LSDs as the base for our agglomeration method.



(a) (b) (c) (d)

Figure 1.4: (a) Ground truth annotations on top of EM data. (b) Boundaries from predicted affinities. (c) Predicted LSDs. There is a clear change of color at all fragment boundaries, while the inner areas of fragments exhibit smooth color gradients. (d) Boundaries using LSDS. Note that the boundaries in (d) do not exhibit the gaps present in (b). Figure by courtesy of Arlo Sheridan.

1.6 Outline

While the hierarchical agglomeration scheme described in Section 1.5 does implicitly use some non-local context by merging edge scores, we believe that explicitly incorporating a wider context from the RAG to the merge function would be very beneficial. Human annotators are able to extract a near-perfect segmentation from a combination of current oversegmentations and raw EM data, so a sophisticated merge function should be able to perform equally well. As describing such a function analytically or by using boolean logic is tedious and dataset-dependent, we suggest to learn it with a supervised machine learning algorithm that makes use of the compressed RAG representation as well as the raw EM data. Variants of Artificial Neural Networks have been very successfully applied to a multitude of tasks in computer vision [41], but only recently performant *Graph Neural Networks* (GNN) have been introduced in the literature that we can use to directly learn from the RAG representation.

The goal of this work is to explore the usage of Graph Neural Networks (GNN) for fragment agglomeration in Region Adjacency Graphs. The first

major challenge is finding a suitable model that is able to capture as much of the available data as possible and that is able to learn effectively. The second major challenge is implementing all necessary steps for learning to agglomerate such that they are scalable and computationally feasible given the available resources.

The remainder of this work is structured in the following way: After presenting other current methods for fragment agglomeration and related variants of Graph Neural Networks, we introduce our formulation of a GNN, called *RagNet*, for fragment agglomeration. The *RagNet* is used to tackle two small-scale synthetic tasks before we apply it to a full-scale EM dataset.

Chapter 2

Related Work

2.1 Agglomeration

Multicut Agglomeration [1] is an another performant approach to fragment agglomeration, apart from the quantile-based hierarchical agglomeration scheme presented in Section 1.5. A multicut is a partition of a weighted graph into a number of components by removing the minimal sum of edge weights. The authors apply a fast iterative approximate algorithm to this NP-hard problem. In [30], this algorithm is adapted to a block-wise hierarchical scheme that is feasible for large-scale Connectomics datasets.

The Region Adjacency Graph can be constructed in various ways. A straightforward approach is to use a summary statistic of the affinities between fragments to initialize the edge scores. For example, our pipeline presented in Section 1.5 uses the maximum affinity between neighboring fragments. Other approaches take a combination of features from the raw data as well as from different preceding steps of the segmentation pipeline for initializing the RAG. For example, in [28, 29] the authors implement active learning for improving the initial edges scores that are based on a multitude of features. Another approach is presented in [2], where both unsupervised and supervised machine learning are explored to learn low dimensional feature representations for fragments that are in turn used to learn local edge decisions.

2.2 Graph Neural Networks

In recent years supervised learning algorithms based on deep neural networks have been successfully applied to data in many different formats, such as images, time series, natural language and website tracking data. While some of these data formats are inherently Euclidian, most others are projected into Euclidian space, e.g. in Natural Language, words are repre-

sented as a low-dimensional vectors in Euclidian space [26]. The reason for projecting data into Euclidian space is that neural network architectures specialized for Euclidian space, most prominently convolutional neural networks (CNN), are surprisingly performant while still being computationally feasible with currently available hardware. Three important properties of Euclidian space that CNNs exploit are the notion of locality and distance, the ability to define translation-invariant operations and compositionality of statistical properties across different scales. The triumph of convolutional neural networks for input data in Euclidian space has sparked efforts to mimic the concept of a convolution in non-Euclidian spaces where it might not be straightforward to define. In particular, there is a field focusing on graph space, point cloud space and manifolds, which is summarized under the term *Geometric Deep Learning* [4]. For this work, we are mostly interested in graph convolutional networks (GCN), an overview of some existing models can be found in [45]. The persistent idea is to aggregate features in local neighborhoods defined by the edges of the graph with a permutation- and node-degree-invariant formulation. Real-world applications of such models include network analysis, various computer vision and graphics tasks, modeling physical systems and medical imaging. There are a few small-scale datasets with corresponding tasks that are being used as benchmarks in the field, e.g. in a rigorous comparative study in [35]. Interestingly, most of these benchmark task are *transductive*, meaning that training is performed on the same graph as inference. In practice, this means that labels for some vertices in the graph are available and all other labels are inferred, similar to a semi-supervised learning task. During completion of this work, only few results for purely inductive tasks such as [14], where inference is performed on graphs not seen during training, have been published.

A few formulations of graph convolutions allow for a combination of graph space with Euclidian space, such as [27, 32]. In [27], Euclidian space is represented by relative coordinates between graph vertices, the so-called *pseudo-coordinates*. They can be seen as features on the graph edges. The neighborhood aggregation in the convolution can now be weighted by some parametrized function of the pseudo-coordinates. In [27], this function is a gaussian mixture model parametrized by mean and covariance for each mixture component, whereas in [8] the function is approximated by spline interpolation with parametrized control points.

Another promising idea was presented as *Graph Attention Network* in [40], which can be used for graphs with an input feature vector for each vertex. The weighting of the neighborhood aggregation is based on a learned function of the two incident vertex features per edge, instead of the pseudo-coordinates used above.

Chapter 3

Methods

3.1 Graph Neural Network

We define a directed, weighted graph as a tuple $\mathcal{G} = ((V, \boldsymbol{X}), (E, \boldsymbol{S}))$, where $V = \{1, 2, \dots\}$ is a set of vertices, $\boldsymbol{X} \in \mathbb{R}^{|V| \times F}$ is a feature matrix containing F features for each vertex, $E \subseteq V \times V$ is a set of edges represented as ordered pairs, and $\boldsymbol{S} \in \mathbb{R}^{|E| \times D}$ holds D-dimensional edge scores. Note that even though a Region Adjacency Graph is undirected, we represent it as a directed graph with the property $(i, j) \in E \iff (j, i) \in E$. However, the edge scores $\boldsymbol{S}_{ij}$ and $\boldsymbol{S}_{ji}$ can differ, e.g., when they contain the relative positioning of the vertices i and j in Euclidian space.

3.1.1 The RagNet layer

Our RAG Attention Graph Network (*RagNet*) is built upon a graph convolution that generalizes those used in *MoNet* [27] and the *Graph Attention Network* [40]. Both are briefly introduced in Section 2.2.

First of all, we define the *one-hop* neighborhood of a vertex i as $\mathcal{N}(i) = \{i\} \cup \{j | (j, i) \in E\}$. A convolution on a vertex i is thus an operation on the features of all vertices in the *one-hop* neighborhood of i:

$$\boldsymbol{X}'_i = \sum_{j \in \mathcal{N}(i)} \alpha_{ji} \boldsymbol{W} \boldsymbol{X}_j \,, \tag{3.1}$$

where $\boldsymbol{W} \in \mathbb{R}^{F' \times F}$ is a learnable parameter matrix and $\alpha_{ji} \in \mathbb{R}$ is a scalar attention coefficient for the directed edge e_{ji} going from vertex j to vertex i. To obtain the attention coefficients α, we first compute the preliminary attention coefficients

$$a_{ji} = \rho(k_\theta(\boldsymbol{X}_j, \boldsymbol{X}_i, \boldsymbol{S}_{ji})) \,, \tag{3.2}$$

where ρ is some non-linear operator and k is called the *kernel* of the convolution, parametrized by θ. We define ρ to be the sigmoid function

$$\rho(x) = \frac{1}{1 + \exp(-x)} \tag{3.3}$$

and k to be a multi-layer perceptron (MLP) [34], using in turn sigmoidal activation functions and batch normalization [17].

We calculate the attention coefficient α_{ji} by normalizing the preliminary attention coefficient a_{ji} over all a's in the *one-hop* neighborhood of vertex i with a *softmax* function:

$$\alpha_{ji} = \frac{\exp(a_{ji})}{\sum_{j'} \exp(a_{j'i})} \,. \tag{3.4}$$

For transforming the features $\boldsymbol{X}^{\ell}$ in layer ℓ to the updated features $\boldsymbol{X}^{\ell+1}$, we perform the convolution defined in Equation 3.1 $C \in \mathbb{N}$ times, apply a non-linear activation function σ and concatenate the resulting feature vectors for each vertex:

$$\boldsymbol{X}_i^{\ell+1} = \Big\Vert_{c=1}^{C} \sigma \left(\sum_{j \in \mathcal{N}(i)} \alpha_{ji}^{\ell,(c)} \boldsymbol{W}^{\ell,(c)} \boldsymbol{X}_j^{\ell} \right) \,. \tag{3.5}$$

After applying L graph convolutional layers to the input graph $\mathcal{G}$ we obtain the transformed feature matrix $\boldsymbol{X}^L$, containing F^L-dimensional feature vectors.

3.1.2 The RagNet loss formulation

We define labels $\boldsymbol{Y} \in \mathbb{R}^{|E| \times \widetilde{E}}$ with $\widetilde{E} \in \mathbb{N}$ to be learned by the *RagNet* via backpropagation of gradients. We can elegantly compute a loss for an edge with a differentiable, similarity-based loss function f that uses the two incident feature vectors from the last *RagNet* layer L. To obtain the total loss, we compute the mean loss over all edges in $\mathcal{G}$:

$$\mathcal{L}_{\text{edges}}(\boldsymbol{X}^L, \boldsymbol{Y}) = \frac{1}{|E|} \sum_{(u,v) \in E} f(\boldsymbol{X}_u^L, \boldsymbol{X}_v^L, \boldsymbol{Y}_{uv}). \tag{3.6}$$

For agglomeration, we define the binary label $\boldsymbol{Y}_{uv} = 1$ for an edge $(u, v) \in E$ that connects vertices that belong to the same object, and conversely $\boldsymbol{Y}_{uv} = 0$ for an edge that connects vertices from different objects. To learn these labels, we would like the *RagNet* to embed the vertices in a latent space that allows us to use an interpretable similarity score for adjacent vertices in $\mathcal{G}$.

Our similarity score of choice is the cosine similarity, which corresponds to measuring the distance of two points on a hypersphere. The so-called *Cosine embedding loss* for the two vertex feature vectors x_u and x_v belonging to the edge $(u, v) \in E$ is defined as

$$f(\boldsymbol{x}_u, \boldsymbol{x}_v, y) = \begin{cases} 1 - \cos(\boldsymbol{x}_u, \boldsymbol{x}_v), & \text{if } y = 1 \\ \max(0, \cos(\boldsymbol{x}_u, \boldsymbol{x}_v) - m), & \text{if } y = 0 \end{cases}, \tag{3.7}$$

where $m \in [0, 1]$ is a margin term to limit the contribution of dissimilar training examples to the overall loss function. It enforces pairwise orthogonality of feature vectors from multiple objects, which is desirable in *Instance Segmentation* problems like ours.

3.2 Learned fragment features

The feature matrix $\boldsymbol{X}$ of the graph $\mathcal{G}$ contains a low-dimensional vector for each RAG fragment. The information content of the feature matrix is crucial for learning, because the vertex feature vectors are actually propagated through the graph by the *RagNet* layers, whereas the edge scores only serve for weighting the contributions of neighbors in a convolution.

We use a deep convolutional neural network inspired by [37] to learn features of RAG fragments directly from 3D image patches. Our approach has been introduced as a *siamese network* in [43]. The idea is to project the inputs into a learned latent space where similar objects are close to each other and dissimilar objects are being pushed away from each other by some loss function. For our application, we consider two neighboring RAG fragments, extract a 3D image patch for each of them and transform them into low-dimensional feature vectors using two identical CNNs with shared weights. The feature vectors are compared using the cosine embedding loss from Equation 3.7. As the ground truth neuron annotations determine for two given RAG fragments whether they belong to the same neuron or not, we can backpropagate the loss gradients to learn a similarity metric for RAG fragments.

Chapter 4

Results

We implement the models presented in Chapter 3 using the Python library *PyTorch Geometric* [9]. This library enhances the popular deep learning library *PyTorch* [31] to handle "irregularly structured input data such as graphs, point clouds or manifolds" with a variety of recently published methods. Furthermore, it provides a flexible, GPU-accelerated neighborhood aggregation scheme for implementing custom definitions for a graph convolution. Interestingly, the GPU acceleration is built upon sparse matrix multiplication, which allows to form mini-batches by simply concatenating the sparse adjacency matrices as well as vertex features and edge scores from multiple graphs of differing sizes.

All performed experiments are logged and easily reproducible via the Python framework *Sacred*[1] [13], a library-agnostic tool for principled running and managing any type of computational experiments.

4.1 Synthetic datasets

Neurons in a specific organism have certain biologically defined structural priors. For example, a strong prior is that each neuron only contains a single soma. A Graph Neural Network should be able to implicitly learn such structural priors and use them in inference. Therefore, we perform multiple experiments in which a GNN based on our *RagNet* layer is trained on synthetic data with some simple structural prior. Our source code[2] is publicly available.

[1] https://github.com/IDSIA/sacred
[2] https://github.com/benjamin9555/gnn_toy_example

4.1.1 Subgraph extraction based on vertex labels

The goal of this experiment is to extract N linear chains of vertices that belong to the same class from a graph in which all pairs of vertices that lie within a certain euclidian distance are connected.

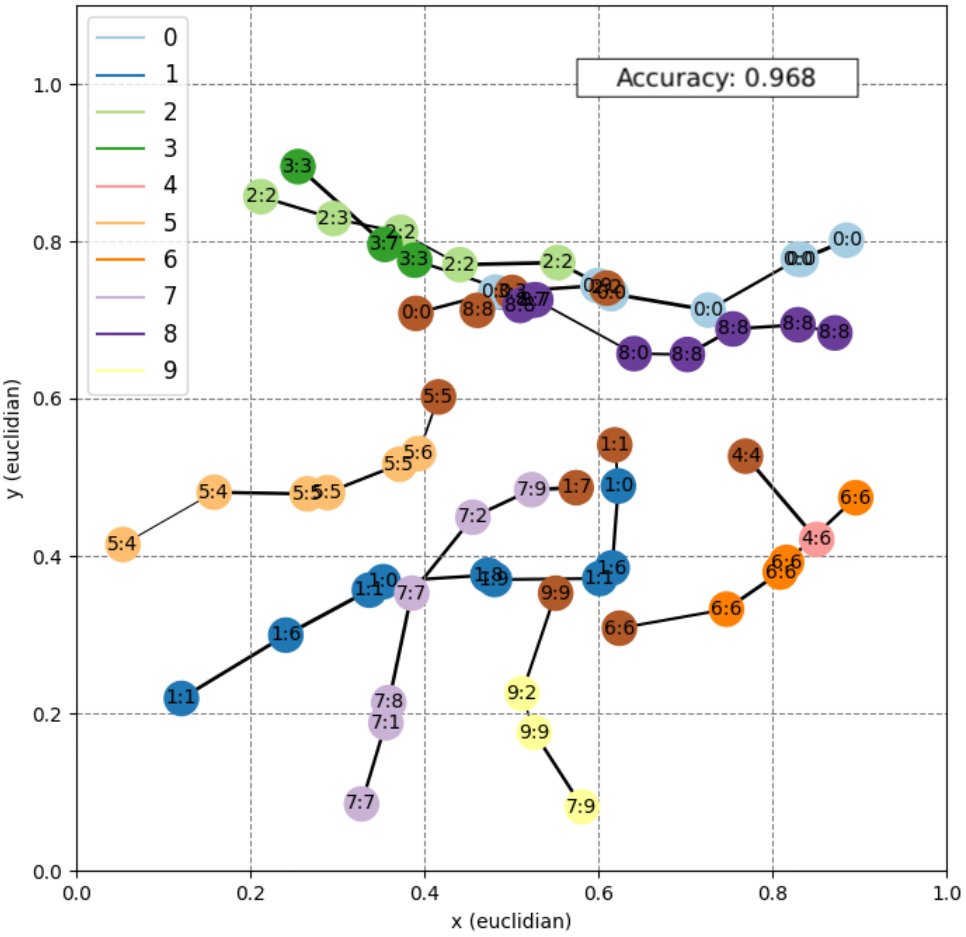


Figure 4.1: Example test output of a *RagNet*, trained to extract linear chains with certain structural priors from a graph. In the input graph, all vertices within a Euclidian distance of at most 0.2 are connected. The few edges visible here represent the ground truth (noise depicted as edge width), and so do the vertex colors. The root vertex of a chain is colored brown. The vertex labels have the format *prediction:noisy input*.

The class membership of each vertex is encoded in two ways. First, the feature matrix $\boldsymbol{X}$ contains noisy, one-hot encoded class membership vectors. Second, all edges $(i, j) \in E$ have a scalar edge score $\boldsymbol{S}_{ij}$ that represents an affinity between the two incident vertices. For the edges that are part of a

linear chain, the affinity is drawn from a negatively skewed beta distribution, while the affinities for all other edges are drawn from a positively skewed beta distribution.

We impose two priors on the linear chains:

- The smallest angle between any two neighboring edges in the chain has a lower bound of 135°, as shown in Figure 4.1.
- The feature matrix $\boldsymbol{X}$ additionally contains a scalar for each vertex. In a chain of vertices, the sequence of these scalars is decreasing.

For this task, instead of having edge-wise labels as defined in Section 3.1.2, the one-hot encoded labels $\boldsymbol{Z} \in \mathbb{Z}^{|V| \times N}$ are placed on the vertices, as is common for most GNN applications in the literature. We define the estimated labels from our *RagNet* as

$$\widehat{\boldsymbol{Z}}_i = \text{softmax}(\boldsymbol{X}_i^L) \qquad \forall i \in V, \tag{4.1}$$

where the number of features in the last *RagNet* layer is $F^L = N$. The total loss is computed as the sum over pairwise losses from $\widehat{\boldsymbol{Z}}$ and $\boldsymbol{Z}$, normalized by the number of vertices in $\mathcal{G}$:

$$\mathcal{L}_{\text{vertices}}(\widehat{\boldsymbol{Z}}, \boldsymbol{Z}) = \frac{1}{|V|} \sum_{i=1}^{|V|} \mathcal{L}(\widehat{\boldsymbol{Z}}_i, \boldsymbol{Z}_i) \quad , \tag{4.2}$$

where $\mathcal{L}$ is the Negative Log-Likelihood loss.

The GNN we use to tackle this task has six *RagNet* layers that each contain an MLP with seven layers for computing the attention coefficients α. It is trained with 1000 randomly created graphs, each containing $N = 10$ different linear chains, and achieves a test accuracy of 94.5% (refer to Figure 4.1).

4.1.2 Subgraph extraction based on edge labels

This task is similar to the one in Section 4.1.1, with the exception of three crucial modifications:

- The feature matrix $\boldsymbol{X}$ does not contain the class membership of vertices. It only contains scalars, which decrease when going along a chain of vertices.
- The labels $\boldsymbol{Y} \in \{0,1\}^{|E|}$ are placed on the edges of $\mathcal{G}$, as originally defined in Section 3.1.2. Edges that connect two vertices from the same chain are labeled with 1, all others with 0.
- Instead of the Cosine Embedding loss outlined in Section 3.1.2, we use an MLP parametrized by θ combined with a squared error loss function to learn the similarity metric:

$$f_\theta(\boldsymbol{x}_u, \boldsymbol{x}_v, y) = \left\| \text{MLP}_\theta(\boldsymbol{x}_u, \boldsymbol{x}_v) - y \right\|^2 . \tag{4.3}$$

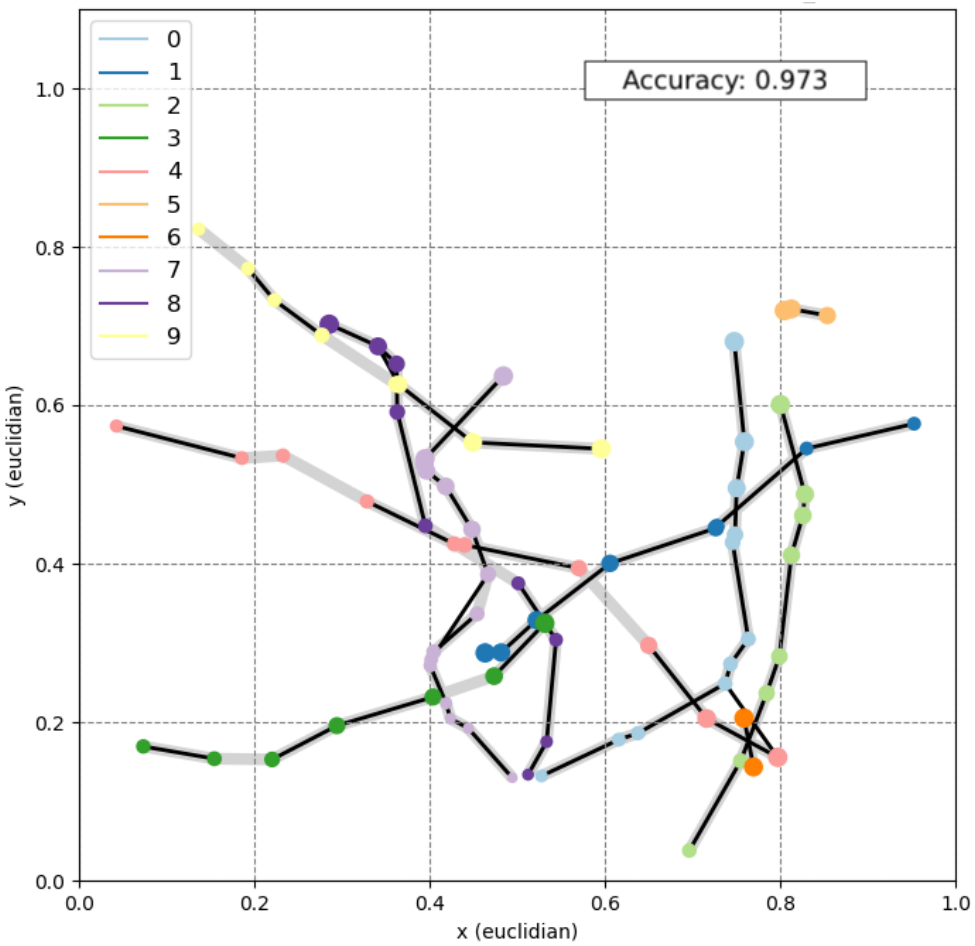


Figure 4.2: Example test output of a *RagNet*, trained to extract linear chains with certain structural priors from a graph. In the input graph, all vertices within a Euclidian distance of at most 0.15 are connected. The vertex colors indicate to which of the ten linear chains each vertex belongs, and the vertex size represents the scalar vertex feature decreasing along chains. The thick gray edges represent the ground truth edges labeled as 1, while the ground truth edges labeled as 0 are not depicted here. The edges that the *RagNet* predicts as connecting linear chains are shown in black.

The *RagNet* we use for this task consists of four *RagNet* layers that each contain an MLP with seven layers for computing the attention coefficients α. Again, it is trained with 1000 randomly created graphs, each containing $N = 10$ different linear chains, and achieves a test accuracy of 95.5% (refer to Figure 4.2).

4.2 Electron microscopy data

4.2.1 RAG ground truth data

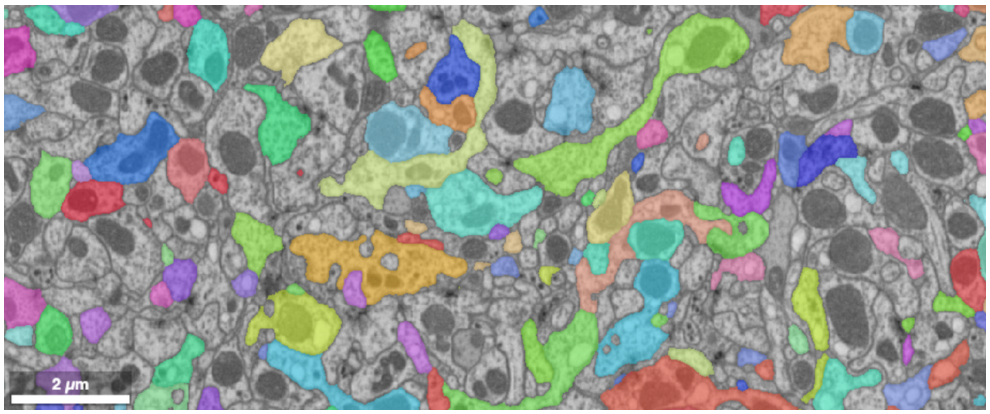


Figure 4.3: Sparse ground truth annotations in the *Hemibrain* dataset

To be able to train a supervised learning approach for agglomerating in the RAG, we need to extract a label for each RAG edge from the provided ground truth segmentations of neurons. Note that these ground truth segmentations are sparse, a substantial part of the neurons in the dataset have not yet been annotated by humans (refer to Figure 4.3). First, we compute the partial overlap with the ground truth segmentations for each fragment in the RAG. Second, if at least 50% of voxels for a given fragment belong to a single neuron, we assign its ID to the fragment, otherwise we assign the fragment to be *background*. Third, these values are then used to assign a ternary adapted ground truth value to each edge in the RAG:

- *merge* if both incident vertices belong to the same annotated neuron.
- *split* if both incident vertices belong to different neurons, or exactly one vertex belongs to the background.
- *unknown* if both incident vertices belong to the background.

We perform the procedure above for three disjunct blocks from the dataset introduced in Section 1.4, all with $8 \times 8 \times 8\,\text{nm}^3$ voxel size. Further details are listed in Table 4.1. The block of size $(21.8\,\mu\text{m})^3$ is used for training the Machine Learning Algorithms, the block of size $(11.8\,\mu\text{m})^3$ for validating and the block of size $(16.8\,\mu\text{m})^3$ for testing.

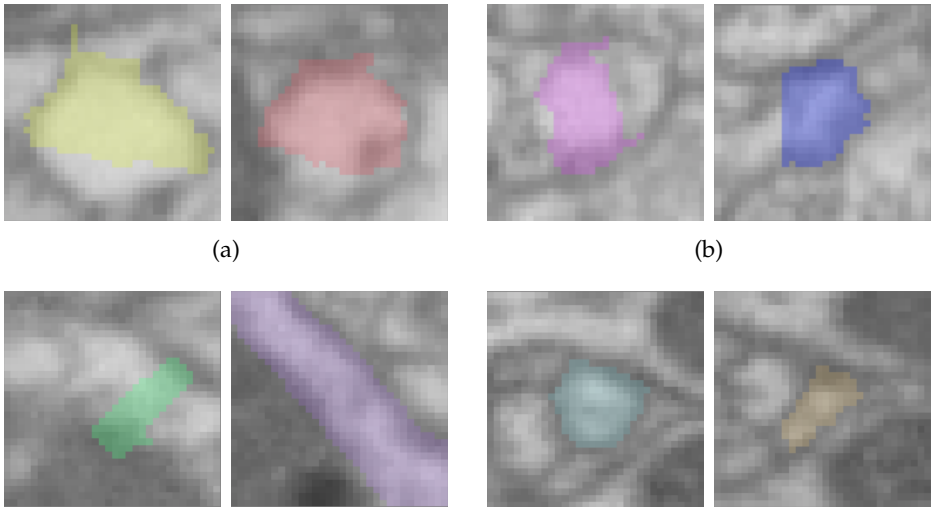
4.2.2 Fragment embeddings from raw EM data

Each fragment in the Region Adjacency Graph corresponds to a subset of voxels from the raw EM data block. While a voxel-wise binary mask already

Table 4.1: Annotated Region Adjacency Graphs from boundary prediction with LSDs. Number of vertices and edges rounded to thousands.

	size	vertices	edges			
			total	merge	split	unknown
training	$(21.8\,\mu m)^3$	1,121K	7,308K	275K	1,597K	5,435K
validation	$(11.8\,\mu m)^3$	135K	876K	59K	286K	530K
test	$(16.8\,\mu m)^3$	429K	2,773K	149K	812K	1,811K

conveys some information about a fragment, like e.g. shape or size, it is clear that the raw data contains additional information. Based on both the binary mask and the raw data, human annotators are able to consistently create a correct segmentation. Therefore it is desirable to create a low-dimensional representation for each fragment that captures as much of the information present in the raw data.



(a) (b) (c) (d)

Figure 4.4: Pairs of EM patches for adjacent RAG fragments used for training the siamese network. Each fragment is marked with a binary mask, depicted in color. The fragment pairs in (a) and (b) belong to the same ground truth neuron and should therefore have similar embeddings. The fragment pairs in (c) and (d) are part of different neurons. Note that the EM data has been downsampled by a factor of 2.

We use a siamese network based on a 3D Convolutional Neural Network as described in Section 3.2 to learn such a low-dimensional representation, commonly referred to as *embedding*. The neural network architecture we

choose is detailed in Table 4.2. The input block for each fragment is a cube of size $512 \times 512 \times 512\,\text{nm}^3$ around its center of mass. Note that we have downsampled the raw data by a factor of 2 in all three dimensions, effectively leading to an input size of $32 \times 32 \times 32$ voxels. Apart from the cube of raw EM data, we create a binary mask of the same size to mark the location of the fragment and use it as a second input channel (refer to Figure 4.4).

Table 4.2: Siamese network architecture. We use $3 \times 3 \times 3$ convolutions and $2 \times 2 \times 2$ Max-Pooling operators. Each convolution is followed by a batch normalization layer [17] and a ReLU function.

Layer	**Feature maps**	**Size per feature map (voxels)**
Input block	2	$32 \times 32 \times 32$
Conv3D	32	$32 \times 32 \times 32$
Conv3D	32	$32 \times 32 \times 32$
MaxPool	32	$16 \times 16 \times 16$
Conv3D	64	$16 \times 16 \times 16$
Conv3D	64	$16 \times 16 \times 16$
MaxPool	64	$8 \times 8 \times 8$
Conv3D	128	$8 \times 8 \times 8$
Conv3D	128	$8 \times 8 \times 8$
MaxPool	128	$4 \times 4 \times 4$
Conv3D	256	$4 \times 4 \times 4$
Conv3D	256	$4 \times 4 \times 4$
MaxPool	256	$2 \times 2 \times 2$
Fully conn.	2048	1
Fully conn.	6	1

Table 4.3: Class-balanced smoothed binary validation accuracy of siamese network after 300,000 iterations, with class separation threshold on cosine similarity set to 0.75. Learning rate at 10^{-4} and embedding dimensionality 9 yields the best result. As accuracy with six-dimensional embeddings is only marginally lower, we choose the smaller embeddings as vertex features for the *RagNet*.

embeddings	lr 10^{-3}	lr 10^{-4}	lr 10^{-5}	lr 10^{-6}
$\mathbb{R}^3$	0.80	0.80	0.79	0.73
$\mathbb{R}^6$	0.81	0.83	0.82	0.75
$\mathbb{R}^9$	0.82	**0.83**	0.80	0.75

We train this network with Adam [21] as the optimization algorithm and

a cosine embedding loss with margin $m = 0.5$. We also lower the margin but do not observe a change in training accuracy. Results on the validation dataset for different learning rates and output vector sizes are summarized in Table 4.3. Data Augmentation such as rotation, elastic deformation and adding voxel intensity noise is performed to avoid overfitting to the training dataset, using the library *gunpowder*[3]. All our source code[4] is publicly available.

4.2.3 RAG agglomeration

The graph $\mathcal{G} = ((V, \boldsymbol{X}), (E, \boldsymbol{S}))$ we use for learning fragment agglomeration consists of the following elements:

- The unique fragment IDs are the set of vertices V.
- The feature matrix $\boldsymbol{X}$ contains six-dimensional learned fragment embeddings (refer to Section 4.2.2).
- For each undirected edge $e = \{u, v\}$ in the RAG, there are two directed edges $e_0 = (u, v)$ and $e_1 = (v, u)$ in the edge set E.
- The edge scores in $\boldsymbol{S}$ are four-dimensional: The *pseudo-coordinates* are modeled by the 3D cartesian vector from vertex u to vertex v, where the absolute location of vertices is represented by the center of mass of the corresponding RAG fragment. Additionally, the scalar RAG merge score (refer to Section 1.5) constitutes the fourth dimension.

A complete RAG for a region of interest in EM data can contain millions of edges (refer to Table 4.1), which is too large for feeding it into an artificial neural network that is trained on a GPU. Therefore, we extract smaller sub-graphs from $\mathcal{G}$ using the euclidian position of vertices. We choose a block size of $1500 \times 1500 \times 1500\,\mathrm{nm}^3$, with an isotropic context of 750 nm surrounding the block. The vertices and edges in the context area are used in the graph convolutional layers to ensure uniform information propagation in the entire block, but the output scores in the context area are disregarded. For training, we extract subgraphs from random locations in the region of interest and augment these subgraphs via rotations and elastic deformations. For inference, we partition the RAG block by block.

The training is carried out on an NVIDIA GeForce RTX 2080 Ti GPU with 11 gigabytes of memory. The available memory puts a limit on the combination of input graph size and network size, which leads to the following choices:

- Each subgraph consists of at most 120000 edges, which typically corresponds to a few thousand vertices.

[3] https://github.com/funkey/gunpowder
[4] https://github.com/benjamin9555/gnn_agglomeration

- The smallest model in our model grid search (refer to Table 4.4) has a total of 12243 trainable parameters, the biggest one has 39133.

The training runs for 1000 iterations of 100 graphs each, which takes between three and seven hours, depending on the exact configuration of the architecture. This roughly corresponds to processing between 4 and 10 graphs per second. In Inference mode, the *RagNet* is able to process approximately 20 graphs per second. Note that these throughput numbers require the graphs to be preprocessed and held in memory.

The used graph neural network consists of multiple stacked *RagNet* layers with batch normalization [17] in between them. This is followed by a cosine embedding loss term with margin $m = 0.5$ (Equation 3.7) for each edge where a ground truth annotation is available. As our training set is not balanced (see Table 4.1), we weight the loss contribution of each class inversely proportional to the class size. We restrict the input of the attention kernels k to the edge scores $\boldsymbol{S}_{ji}$ and omit the vertex features $\boldsymbol{X}_j$ and $\boldsymbol{X}_i$ for simplicity.

Model grid search

We train 36 different models and compare the class-balanced accuracy on the test set in Table 4.4. The learning rate in Table 4.4 is used the initialize the Adam optimizer [21].

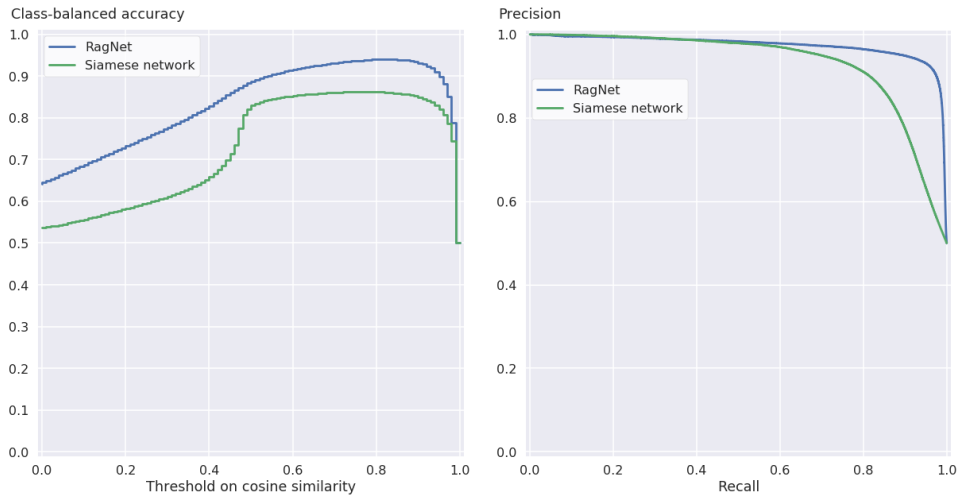
Table 4.4: Class-balanced binary test accuracy of *RagNet*. Accuracy generally improves when adding more *RagNet* layers. The learning rate (lr) and the number of attention layers do not do not exhibit a clear trend. Architecture details in Appendix A.

		3 *RagNet* layers	4 *RagNet* layers	5 *RagNet* layers	6 *RagNet* layers
3 attention layers	lr $1 * 10^{-3}$	0.919	0.913	0.916	**0.934**
	lr $5 * 10^{-4}$	0.829	0.924	0.927	0.925
	lr $1 * 10^{-4}$	0.914	0.915	0.923	0.928
4 attention layers	lr $1 * 10^{-3}$	0.821	0.920	0.921	0.925
	lr $5 * 10^{-4}$	0.913	0.920	0.928	0.922
	lr $1 * 10^{-4}$	0.914	**0.924**	0.920	0.922
5 attention layers	lr $1 * 10^{-3}$	0.918	0.919	0.923	0.931
	lr $5 * 10^{-4}$	**0.921**	0.917	**0.933**	0.924
	lr $1 * 10^{-4}$	0.909	0.914	0.928	0.930

Comparison to fragment embeddings

We compare the performance of our best *RagNet* to the performance achieved by solely the fragment embeddings from Section 4.2.2 in Figure 4.5

and observe a clear improvement. Moreover, we qualitatively compare the fragment embeddings generated by the siamese network to the low-dimensional vertex representations in the *RagNet* right before we apply the cosine embedding loss. To do so, we use Principle Component Analysis on the six- and respectively eight-dimensional vectors and visualize the first three principal components as RGB values in Figure 4.6.



(a) (b)

Figure 4.5: (a) Class-balanced binary validation accuracy of siamese network and *RagNet*. (b) Class-balanced Precision-Recall curves for splits, for both siamese network and *RagNet*. The *RagNet* clearly improves upon the siamese network on both metrics.

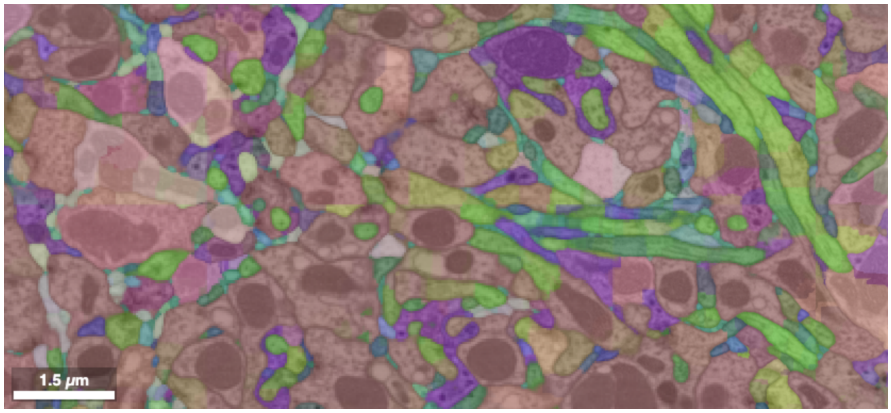
Comparison to state of the art

Furthermore, we compare segmentations produced by our best *RagNet* against segmentations by state-of-the-art methods that use hierarchical agglomeration using the Variation of Information (VOI) metric, a commonly used metric in Connectomics.

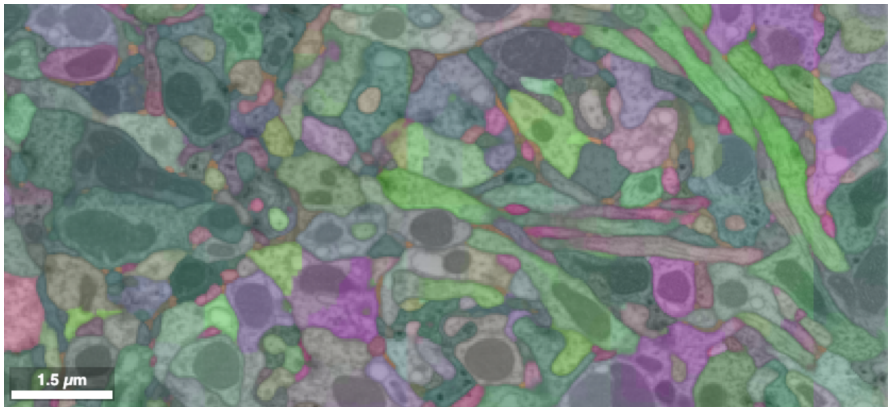
The variation of information (VOI) is a distance measure between two partitions of a set A. The two random variables X and Y that correspond to the two partitions are used to calculate the conditional entropies

$$H(Y|X) = -\sum_{x \in X} p(x) \sum_{y \in Y} p(y|x) \log p(y|x) \tag{4.4}$$

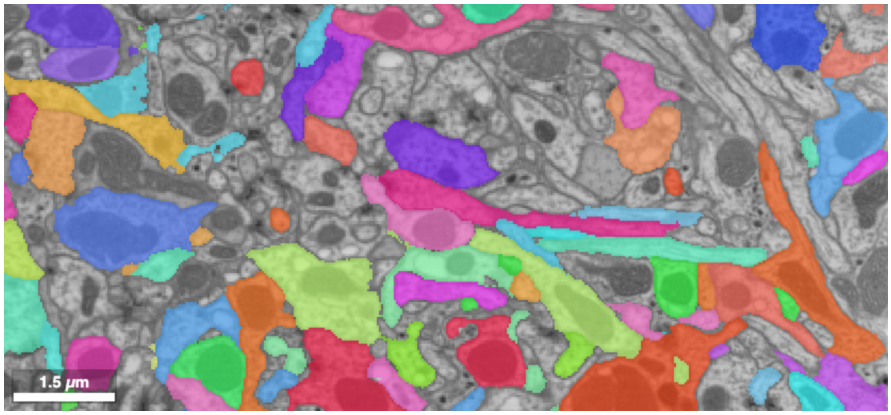
$$H(X|Y) = -\sum_{y \in Y} p(y) \sum_{x \in X} p(x|y) \log p(x|y), \tag{4.5}$$



(a) Embeddings by siamese network



(b) Embeddings by *RagNet*



(c) Ground truth annotations

Figure 4.6: Example of generated embeddings

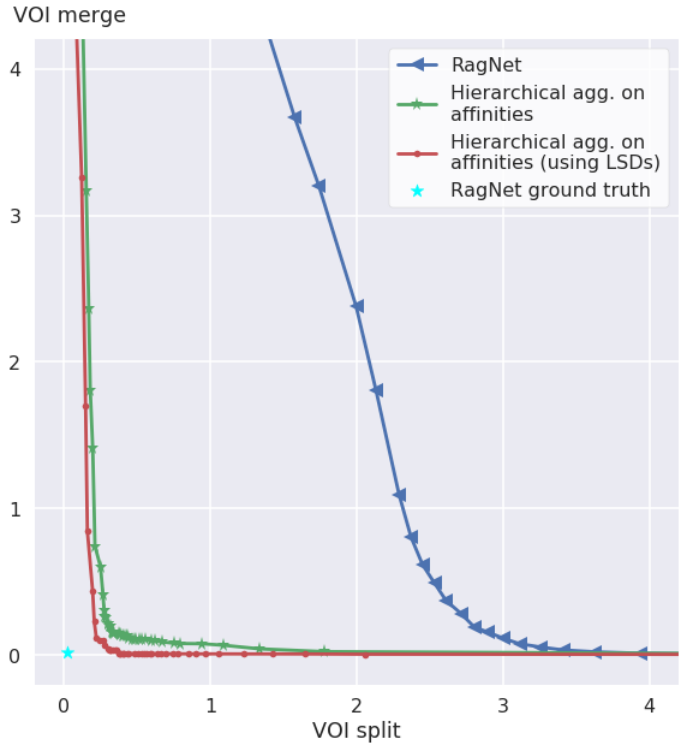


Figure 4.7: Variation of Information of the best *RagNet* compared to hierarchical agglomeration approaches, lower is better. We also show the ground truth generated in Section 4.2.1, which is the best VOI the *RagNet* could potentially achieve.

which are summed up to

$$\text{VOI} = H(Y|X) + H(X|Y). \tag{4.6}$$

In neuron reconstruction, the two VOI summands measure false splits and false merges of a segmentation compared to the ground truth. Assume that X corresponds to the ground truth and Y to the segmentation, then $H(Y|X)$ quantifies false splits and $H(X|Y)$ quantifies false merges.

We compare our Graph Neural Network against two methods using hierarchical agglomeration in Figure 4.7. In the first one, the boundary prediction is done with regular affinities (refer to Section 1.5), the second one also uses local shape descriptors. It is clear that the *RagNet* is not able to compete with these approaches on VOI.

We also compare the *RagNet* to these two methods in terms of edge-wise error rate (1 - accuracy) and put it in context with corresponding VOI sums in Figure 4.8. As seen before in Figure 4.7, the VOI sums of the hierarchical

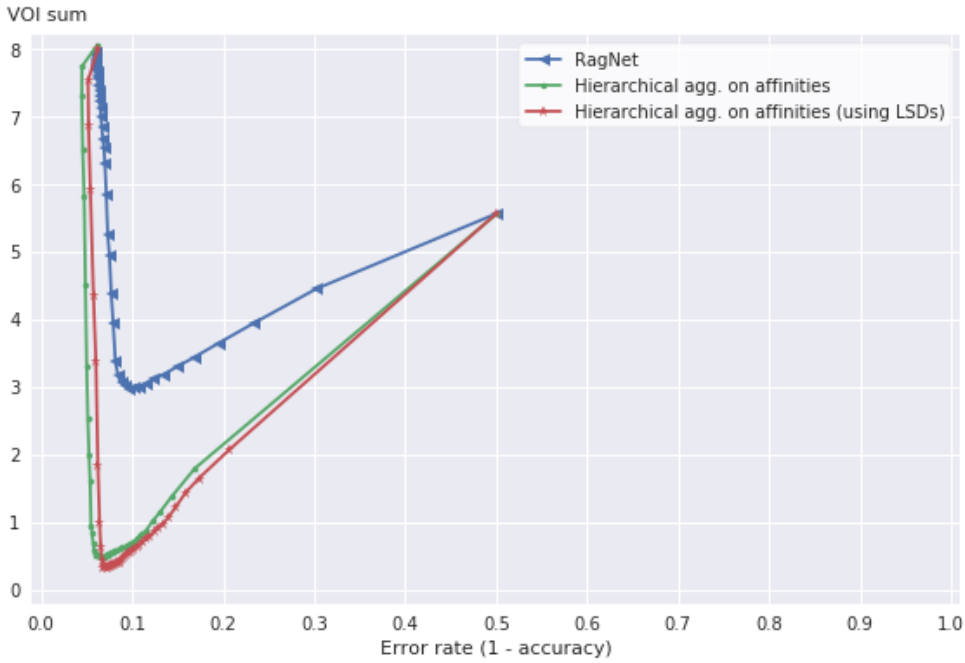
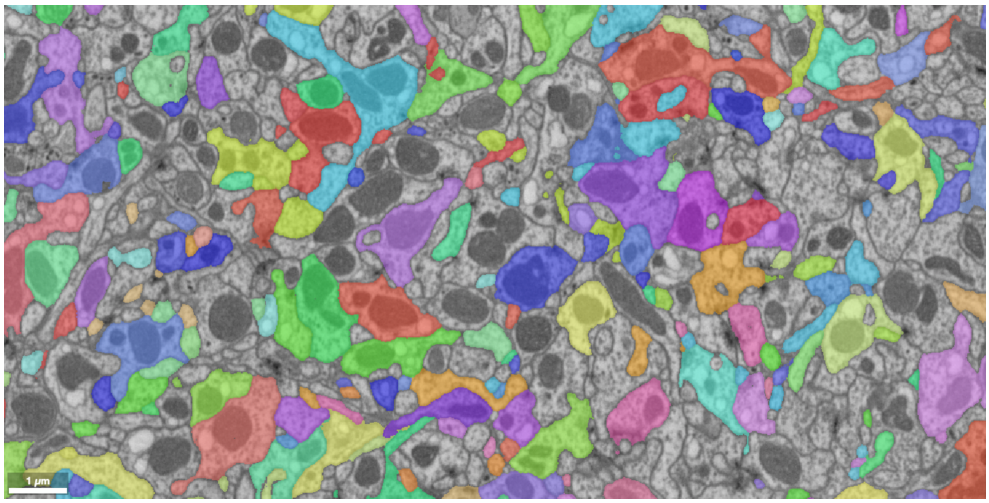


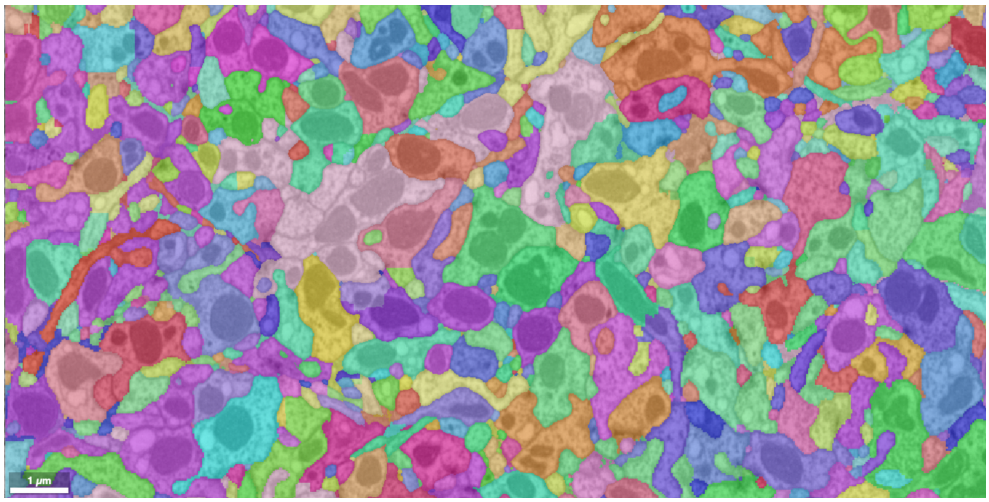
Figure 4.8: Correspondence of binary error rate (1 minus class-balanced accuracy) and VOI sum (split + merge). While the hierarchical agglomeration is far better than the *RagNet* measured by VOI, the error rate at the elbows only differs by a few percentage points.

agglomeration methods are clearly better than the *RagNet*. Interestingly, the error rates at the elbows of the curves only differ by a few percentage points, indicating that the *RagNet* can potentially compete with the two state-of-the-art methods. As the *RagNet* was only trained with a non-structural, edge-wise loss, it seems to lack the guidance to make the necessary tradeoffs to achieve a better VOI score.

Qualitative results



(a) Sparse ground truth



(b) Segments extracted by *RagNet*

Figure 4.9: In a 2D representation, the extracted segments locally correspond to the sparse ground truth annotations. However, there are some clear false merges visible.

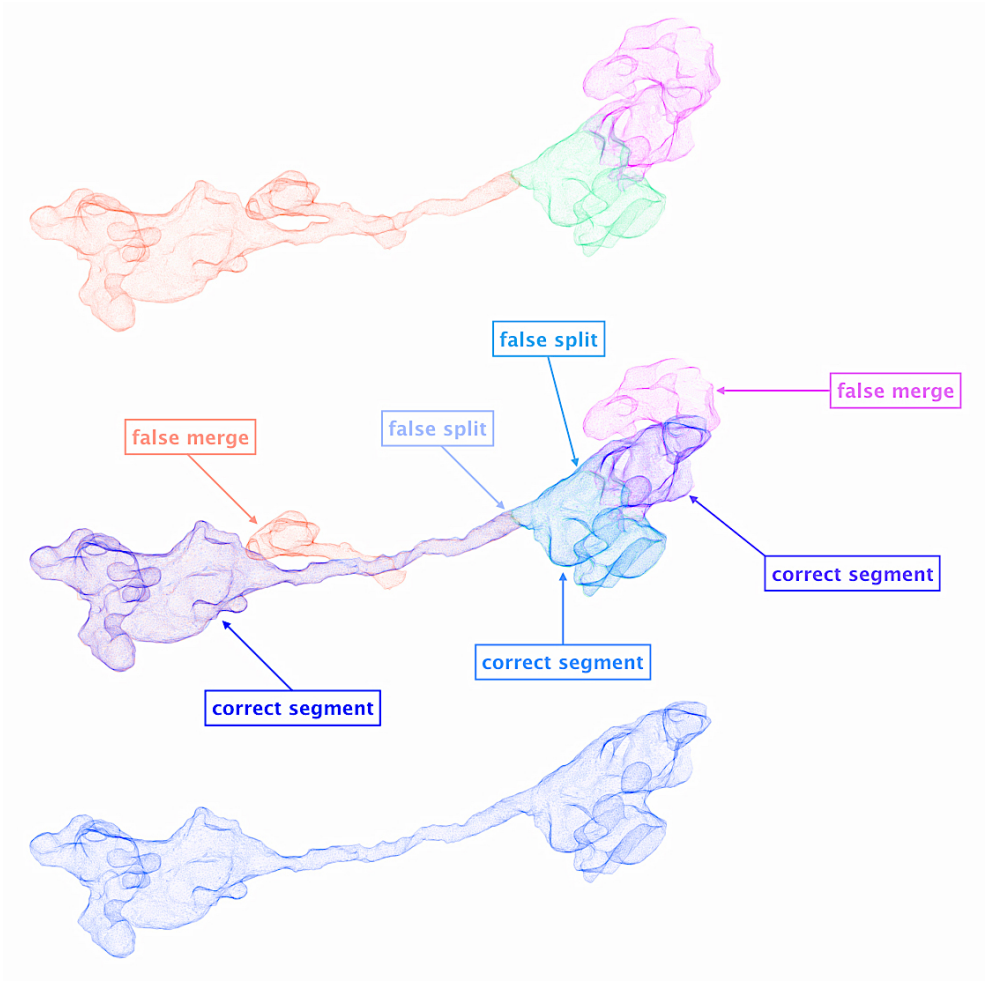


Figure 4.10: Example reconstruction of a small *Drosophila* neuron. Our segmentation with *RagNet* at the top, ground truth at the bottom, the overlay in the center.

Chapter 5

Discussion

In this master thesis, we developed a proof of concept that Graph Neural Networks can be used to learn fragment agglomeration in Region Adjacency Graphs and implemented our approach as part of an existing large-scale neuron reconstruction pipeline. Our formulation of a GNN is able to clearly improve upon fragment feature representations that are learned from raw electron microscopy data without any structural context.

However, our results are not yet comparable to state-of-the-art agglomeration algorithms. Generally, it has to be shown that GNNs are able to perform well on large scale inductive inference tasks. The field has not yet converged to using a few performant network architectures, unlike more prominent deep learning fields such as Image Classification (VGG [37]), Object Segmentation (U-Net) or Natural Language Understanding (LSTM [16]). The consequent development of our own model *RagNet* was empirically supported by promising results on various tasks. Having said this, it is necessary to gain a better understanding of both our model and generally GNNs. One future step is visualizing the kernels of the graph convolutions, as these might contain indicative information about the message flow in the graph. Furthermore, a rigorous stability analysis is required to scale our model to depths beyond the few layers we have used in this work, as we have already experienced vanishing gradients at relatively small size.

An obvious next step to improve the GNN performance on the particular task of fragment agglomeration is to introduce a structured loss, similar to MALIS for affinity prediction, in order to get stronger gradients for errors that have a large impact on the final segmentation.

It would be interesting to see whether the improvement of the GNN over agglomeration on local fragment features also holds true on anisotropic data such as the FAFB dataset. Furthermore, to process a dataset of that size, it is also necessary to overcome the current minor scaling issues, mostly by

adapting the block-wise processing to the needs of a GNN. Conceptually, inference is fully scalable. Parallelized to, e.g., 8 GPUs, as commonly available on a single cluster node, inference for the entire FAFB dataset would take roughly two days with the current implementation.

Another interesting lever to improve learned agglomeration is extracting better features from the raw EM data. Two things come to mind for this. First, feature extraction could be made size-invariant, as fragments sizes differ largely. The current embeddings from fixed-size patches exhibit deficiencies for larger fragments, due to our choice of not neglecting the small fragments and choosing the patch specifications based on them. Second, it is desirable to connect the feature extraction network to the GNN to create an end-to-end learned approach for agglomeration. Bear in mind that this could potentially amplify the stability issues mentioned above.

In conclusion, we believe that Graph Neural Networks can become a beneficial step in automated neuron reconstruction if the Connectomics community, at best jointly with the Geometric Deep Learning community, decides to deepen the effort on analyzing and understanding them.

Appendix A

Appendix

Table A.1: Detailed setups of attention kernel multi-layer perceptrons

	input size	layer sizes	number of parameters
3 attention layers	4	32, 16, 1	656
4 attention layers	4	32, 16, 8, 1	776
5 attention layers	4	32, 16, 8, 4, 1	804

Table A.2: Detailed *RagNet* setups. To obtain the number of attention kernels per layer, divide the respective number of feature maps by 8.

	number of feature maps
3 *RagNet* layers	64, 32, 8
4 *RagNet* layers	64, 32, 16, 8
5 *RagNet* layers	64, 64, 32, 16, 8
6 *RagNet* layers	64, 64, 64, 32, 16, 8

List of Figures

List of Tables

Bibliography

[1] Thorsten Beier, Constantin Pape, Nasim Rahaman, Timo Prange, Stuart Berg, Davi D Bock, Albert Cardona, Graham W Knott, Stephen M Plaza, Louis K Scheffer, et al. Multicut brings automated neurite segmentation closer to human performance. *Nature methods*, 14(2):101, 2017.

[2] John A Bogovic, Gary B Huang, and Viren Jain. Learned versus hand-designed feature representations for 3d agglomeration. *arXiv preprint arXiv:1312.6159*, 2013.

[3] Kevin Briggman, Winfried Denk, Sebastian Seung, Moritz N Helmstaedter, and Srinivas C Turaga. Maximin affinity learning of image segmentation. In *Advances in Neural Information Processing Systems*, pages 1865–1873, 2009.

[4] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[5] Julia Buhmann, Renate Krause, Rodrigo Ceballos Lentini, Nils Eckstein, Matthew Cook, Srinivas Turaga, and Jan Funke. Synaptic partner prediction from point annotations in insect brains. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 309–316. Springer, 2018.

[6] Özgün Çiçek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d U-net: learning dense volumetric segmentation from sparse annotation. In *International conference on medical image computing and computer-assisted intervention*, pages 424–432. Springer, 2016.

[7] Juan A De Carlos and José Borrell. A historical reflection of the contributions of Cajal and Golgi to the foundations of neuroscience. *Brain research reviews*, 55(1):8–16, 2007.

[8] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 869–877, 2018.

[9] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[10] HHMI Janelia FlyEM project. Drosophila *hemibrain* dataset imaged with FIB-SEM (in preparation).

[11] Jan Funke. *Automatic Neuron Reconstruction from Anisotropic Electron Microscopy Volumes*. PhD thesis, ETH Zurich, 2014.

[12] Jan Funke, Fabian Tschopp, William Grisaitis, Arlo Sheridan, Chandan Singh, Stephan Saalfeld, and Srinivas C Turaga. Large scale image segmentation with structured loss based deep learning for connectome reconstruction. *IEEE transactions on pattern analysis and machine intelligence*, 41(7):1669–1680, 2018.

[13] Klaus Greff, Aaron Klein, Martin Chovanec, Frank Hutter, and Jürgen Schmidhuber. The sacred infrastructure for computational research. In *Proceedings of the Python in Science Conferences-SciPy Conferences*, 2017.

[14] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[15] Kenneth J Hayworth, C Shan Xu, Zhiyuan Lu, Graham W Knott, Richard D Fetter, Juan Carlos Tapia, Jeff W Lichtman, and Harald F Hess. Ultrastructurally smooth thick partitioning and volume stitching for large-scale connectomics. *Nature methods*, 12(4):319, 2015.

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

[17] Sergey Ioffe and Christian Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[18] Michał Januszewski, Jörgen Kornfeld, Peter H Li, Art Pope, Tim Blakely, Larry Lindsey, Jeremy Maitin-Shepard, Mike Tyka, Winfried Denk, and

Viren Jain. High-precision automated reconstruction of neurons with flood-filling networks. *Nature methods*, 15(8):605, 2018.

[19] Verena Kaynig, Amelio Vazquez-Reina, Seymour Knowles-Barley, Mike Roberts, Thouis R Jones, Narayanan Kasthuri, Eric Miller, Jeff Lichtman, and Hanspeter Pfister. Large-scale automatic reconstruction of neuronal processes from electron microscopy images. *Medical image analysis*, 22(1):77–88, 2015.

[20] Khaled Khairy, Gennady Denisov, and Stephan Saalfeld. Joint deformable registration of large EM image volumes: A matrix solver approach. *arXiv preprint arXiv:1804.10019*, 2018.

[21] Diederik P Kingma and Jimmy Ba. Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[22] Jörgen Kornfeld and Winfried Denk. Progress and remaining challenges in high-throughput volume electron microscopy. *Current opinion in neurobiology*, 50:261–267, 2018.

[23] Kisuk Lee, Nicholas Turner, Thomas Macrina, Jingpeng Wu, Ran Lu, and H Sebastian Seung. Convolutional nets for reconstructing neural circuits from brain images acquired by serial section electron microscopy. *Current opinion in neurobiology*, 55:188–198, 2019.

[24] Kisuk Lee, Jonathan Zung, Peter Li, Viren Jain, and H Sebastian Seung. Superhuman accuracy on the SNEMI3D connectomics challenge. *arXiv preprint arXiv:1706.00120*, 2017.

[25] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[26] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[27] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017.

[28] Juan Nunez-Iglesias, Ryan Kennedy, Toufiq Parag, Jianbo Shi, and Dmitri B Chklovskii. Machine learning of hierarchical clustering to segment 2d and 3d images. *PloS one*, 8(8):e71715, 2013.

[29] Juan Nunez-Iglesias, Ryan Kennedy, Stephen M Plaza, Anirban Chakraborty, and William T Katz. Graph-based active learning of agglomeration (gala): a python library to segment 2d and 3d neuroimages. *Frontiers in neuroinformatics*, 8:34, 2014.

[30] Constantin Pape, Thorsten Beier, Peter Li, Viren Jain, Davi D Bock, and Anna Kreshuk. Solving large multicut problems for connectomics via domain decomposition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1–10, 2017.

[31] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[32] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017.

[33] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[34] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[35] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[36] Arlo Sheridan et al. Local shape descriptors for neuron segmentation (in preparation).

[37] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[38] Olaf Sporns, Giulio Tononi, and Rolf Kötter. The human connectome: a structural description of the human brain. *PLoS computational biology*, 1(4):e42, 2005.

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

[40] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[41] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: a brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[42] John G White, Eileen Southgate, J Nichol Thomson, and Sydney Brenner. The structure of the nervous system of the nematode caenorhabditis elegans. *Philos Trans R Soc Lond B Biol Sci*, 314(1165):1–340, 1986.

[43] Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4353–4361, 2015.

[44] Zhihao Zheng, J Scott Lauritzen, Eric Perlman, Camenzind G Robinson, Matthew Nichols, Daniel Milkie, Omar Torrens, John Price, Corey B Fisher, Nadiya Sharifi, et al. A complete electron microscopy volume of the brain of adult drosophila melanogaster. *Cell*, 174(3):730–743, 2018.

[45] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Learning to Agglomerate in Region Adjacency Graphs for Electron Microscopy Neuron Segmentation using Graph Neural networks

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Gallusser	Benjamin

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Ashburn, VA, USA September 15, 2019	B. Gallusser

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.