Technische Universität München
Fakultät für Physik

Abschlussarbeit im Bachelorstudiengang Physik

Analysis of Neuronal Morphology Using Semantic Segmentation of Point Clouds

Analyse neuronaler Morphologie mittels semantischer Segmentierung von Punktwolken

Jonathan Klimesch

31.03.2020

Max-Planck-Institut für Neurobiologie

Contents

Chapter 1

Introduction

1.1 Structural Neurobiology

1.1.1 Structure of neurons

Neurons are responsible for signal reception, processing and transmission in the nervous system of numerous species. They have varying morphologies which correlate with their functionality and behaviour. However, almost all vertebrate neurons essentially consist of three components, a soma which gives rise to processes called dendrites and axons. The shape of the soma, the cell body containing the nucleus, varies together with the overall cell morphology. Possible shapes are pyramidal, oval, granular, cerebellar, stellar or diffuse [12]. The soma is responsible for maintaining the neuron structure and function. It is therefore equipped with all necessary organelles like the nucleus, the Golgi apparatus, mitochondria, polysomes, cytoskeletal elements and lysosomes to produce macromolecules or to recycle other important elements of the cell. Besides ensuring the cell stability, the soma also processes and dispatches incoming signals. These signals are either transmitted by other cells that connect directly to the soma or are forwarded from the surrounding via dendrites. Since signals must often be transported over long distances, a neuron takes on a rather elongated shape and forms extensions that spread away from the soma, namely axons and dendrites [24]. Fig. 1.1 shows a schematic drawing of a neuron with its main components.

An axon has a main branch, which can sometimes reach lengths of more than one meter. At the end of this branch it builds ramifications, so-called telodendria, that allow multiple connections to other neurons. This way, each signal transmitted by the soma can be forwared to a large number of other neurons. These signals are then detected by the second type of cell extensions, the dendrites. Neuron branches have a widely ramified structure and cover a large area to connect to as many other neurons as possible. Incoming signals are transmitted along the dendritic shaft to the soma, from where they

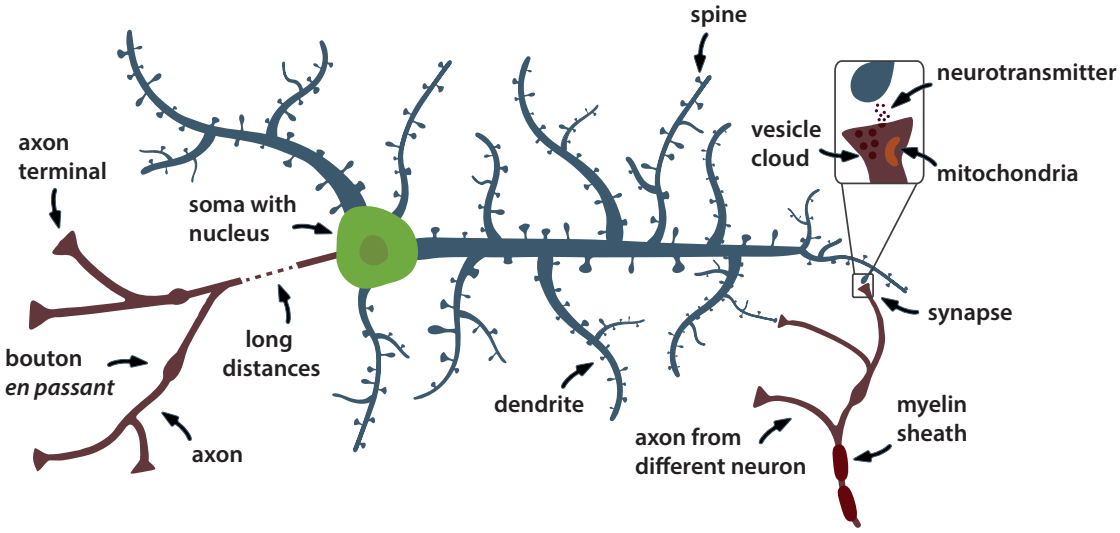

Figure 1.1: Schematic drawing of a neuron which consists of mainly 3 parts: The soma, axons and dendrites. Axons have presynaptic elements called axon terminals or boutons, dendrites have postsynaptic spines, which can have various morphologies.

are forwarded again to the axons [4]. Morphologically, axons and dendrites differ in many ways. Except from some local thickenings and additional layers of myelin, axons have a nearly uniform diameter. In contrast, the diameter of dendrites decreases along the ramifications. Dendrites have more acute angles between their branches and show different ultrastructural characteristics. The surface of axons is often smooth without many protuberances, while dendrite surfaces have a rather high branch density due to dendritic arbors with small branches called spines [12].

The human brain contains more than 10^{13} of these spines, which are mainly located on dendrites, but also partly on the soma [28]. From a morphological point of view, spines can be divided into several categories. Mushroom spines consist of a large head connected to the dendritic shaft by a thin neck. Thin spines have a long neck, but only a small head. Stubby spines have a large head, which is attached directly to the shaft without a neck. The filopodium category describes protuberances with hair-like morphology, very thin necks that protrude from the dendritic shaft without a head and are mainly found during the developmental phase of a dendrite branch. Spines play an essential role in signal transmission between cells and form on the postsynaptic part of a synapse [28].

1.1.2 Synapses and cellular organelles

Synapses consist of a presynaptic part, which is usually found on axons, and a postsynaptic part, which is located on dendrite, spine or soma. The presynaptic parts are located on synaptic boutons or axon terminals at the ends of axonal branches. It is also possible that an axon builds contact sites in form of thickenings along its branches which are then called boutons *en passant*. Morphologically, synapses show different ultrastructural characteristics and have a narrow gap between their two parts, the synaptic cleft. This gap forms an electrical insulation, so that signals must be transmitted by chemical messengers. In addition to the messenger based exchange, there are also electrical synapses in which the ions flow directly from one cell to the other.

The presynaptic element, e.g. an axon terminal, is characterized by different cellular organelles. Often, they are equipped with multiple mitochondria, organelles in the cytoplasm which have the size of small bacteria and are mainly responsible for energy production [12].

The chemical messengers used for signal transmission are stored in small secretory vesicles, another type of cell organelle that is essential for synapses. Each of these membrane-enclosed containers with a diameter of about 50 nm holds multiple neurotransmitters, chemical signal substances, which are released into the synaptic cleft by fusion with the terminal membrane in a process called exocytosis [12]. Depending on the neurotransmitters and the transmitter-gated ion channels, chemical synapses can have both excitatory and inhibitory effects. The type of synapse depends on the location, the selectivity of the channels and the ion conditions [4].

1.1.3 Connectomics

The human brain consists of about 10^{11} neurons, each of which is connected to many other neurons [24]. In order to understand how this complex machine works, one approach is to extract the structure of these neuron connections. The field of research that deals with this reconstruction is called Connectomics and aims to create complete circuit diagrams of different brains and brain regions. By creating these diagrams for sub-volumes of brain regions, entire regions or even brains, it may for example be possible to examine the process of storing memories and experience. By comparing the circuit diagrams of healthy and diseased brains, the physical causes of psychiatric diseases could be analyzed in more depth. The differences in the neural circuits of young and old brains, could provide hints about the effects of aging. The comparison of connectomes of a human and another primate could provide insights into the causes of intelligence. The

knowledge of these connectomes could therefore be an essential step towards a detailed understanding of the brain [25].

However, the creation and analysis of more complex connectomes is technically challenging. So far, only three complete data sets of brains of different sizes are available: The structure of the nervous system of the Caenorhabditis elegans with 302 neurons, the nervous system of the larval fly with about 10,000 neurons and the brain of Drosophila melanogaster with about 100,000 neurons [18].

Given the large number of neurons, both the acquisition and the analysis of the data required to reconstruct the neural wiring must be fast and yet very accurate. The next section summarizes the current progress in volume electron microscopy (VEM) as the state-of-the-art technique for data acquisition in Connectomics. Even small data sets (around $0.002\,\mathrm{mm}^3$) would require an immense amount of time in order to annotate them properly. Therefore, algorithms have been developed to process data sets automatically. Recent approaches use different types of artificial neural networks to reconstruct all organelles in the data set and to analyse their components [9]. Later sections will discuss the foundations of these artificial neural networks and their use for semantic segmentation which is the core topic of this thesis.

1.2 EM data acquisition

Sufficient resolution of densely packed dendritic and axonal processes in neural circuits can be reached by electron microscopes (EM). With transmission electron microscopes (TEM) the sample is illuminated by a broad beam of electrons and the contrast in the images is generated by the elastic scattering of electrons in high density regions of the tissue. In order to create actual 3D datasets with volume electron microscopy (VEM), ultrathin sections are cut from a block of tissue and are aligned for data acquisition [17]. These sections can then be analyzed using serial-section TEM (SSTEM), a technique which was used for all of the three previously mentioned connectome datasets [18]. However, the registration and alignment of the serial sections in this method is problematic, as the serial sectioning introduces tears, scratches and distortions that require complex corrections [38].

An alternative are scanning electron microscopes (SEM), which use a tightly focused beam of electrons to raster-scan over the tissue. By capturing secondary or backscattered electrons from below the surface of the tissue, this method can be used to visualize three dimensional surfaces. The alignment problem of SSTEM can be solved by serially

sectioning the tissue with a diamond microtome, where the knife is moved, but the tissue is fixed. This method is called serial block-face electron microscopy (SBEM), as it cuts slices of the tissue and scans the remaining block face iteratively using SEM. The created surface images can then be stacked together to represent an actual 3D dataset [8].

Fig. 1.2 shows images from the EM data set used in this thesis (named j0126). It was acquired by Jörgen Kornfeld using SBEM and has an extent of $96 \times 98 \times 114\,\mu m$ with an xyz-resolution of $9 \times 9 \times 20\,nm^3$. The tissue was extracted from the Area X of a zebra finch, which is located within the songbird basal ganglia that is part of a neural song learning circuit [35].

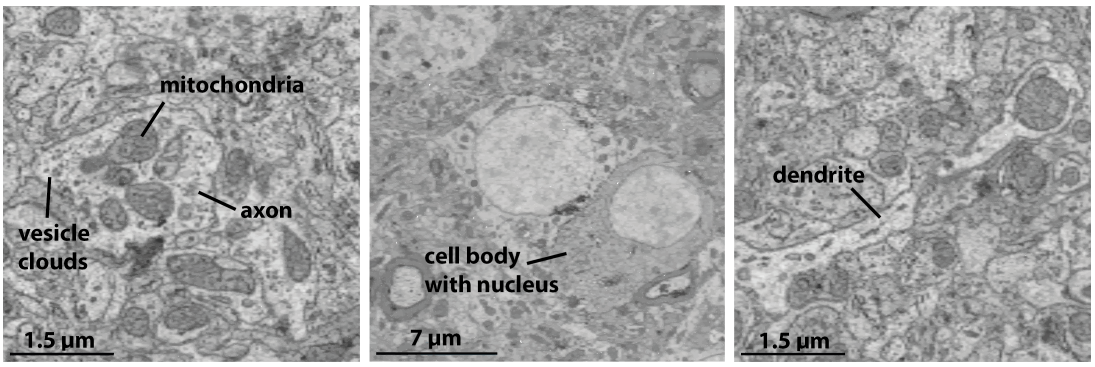

Figure 1.2: EM data acquired using serial block-face electron microscopy (SBEM).

The SBEM method can be further enhanced by replacing the microtome with a focused ion beam (FIB) which can remove material from the block face in increments of as low as 2 nm, whereas the SBEM method is limited to around 20 nm [18]. The overall signal of the backscattered electrons can be enhanced by proper staining techniques which are applied prior to the data acquisition [8].

The field-of-view of the FIB-SBEM method is limited to a few tens of microns, causing the data acquisition of a whole mouse brain to last for an unreasonable amount of time. However, a recent technique used a lubricated and heated diamond knife to cut epoxy-embedded heavy metal-stained brain-tissue blocks into relatively thick (10 to 30 µm) samples with only minimal data loss at the interfaces. Combining this cutting technique with multiple FIB-SBEM machines could parallelize the data acquisition and therefore reduce the duration of it by a large scale [18].

1.3 Artificial Neural Networks

This section discusses the foundations of artificial neural networks, convolutional neural networks and their extension to three spatial dimensions. The section is mainly based on the descriptions in Michael Nielsen's book *Neural Networks and Deep Learning* [27].

1.3.1 Neural Networks

Depending on parameters like the type of neurotransmitters and ion channels, synapses can have excitatory, inhibitory or modulatory effects. The strength of a stimulation is determined by weighting the different potentials at the soma and is then used to set the firing rate of the neuron [24].

These neurobiological concepts were used by researchers in the field of computational neuroscience to create a mathematical abstraction of biological nerve cells, so-called artificial neurons. The first versions of such neurons have their origin as early as 1943, developed by Warren McCulloch and Walter Pitts. Later, Frank Rosenblatt developed the so-called perceptron, which is the predecessor of the modern artificial neuron.

An artificial neuron is a system with inputs $x_1, x_2, ...$, which are associated with weights $w_1, w_2, ...$ The inputs and weights can be compared to incoming signals at synapses and their excitatory or inhibitory properties. The artificial neuron processes these inputs by the formula

$$h(\vec{x}) = g(\sum_j w_j x_j + b) \tag{1.1}$$

where $\vec{x}$ is the vector of inputs and b is the bias, a property of the neuron which indicates its likelihood to fire even in the absence of other inputs. The neuron's inputs are then transformed by the so-called activation function $g(\cdot)$. Common activation functions are for example the sigmoid function or a rectified linear unit (ReLU, [26]), which extend the system with non-linearities [10]. The output of the artificial neuron is given by $h(\vec{x})$, which can be compared to the biological firing rate in a short time window.

Combining multiple of such artificial neurons by using the output of one neuron as the input of multiple other neurons results in an artificial neural network. The neurons in the first layer of such a network are called input neurons, the last layer contains the output neurons. In between are so-called hidden layers of neurons. If there are two or

more hidden layers, the network is called deep, which is the origin of the name deep learning. Each of the neurons in the first hidden layer has all input neurons as its inputs and forwards its outputs to all neurons in the next hidden layer or the output layer. Such a network, in which all neurons of neighboring layers are connected is called a fully connected neural network.

Given a sufficiently large number of training examples, tuples of inputs x_j and outputs y_j, such a network can approximate any function which maps the inputs to the outputs by adjusting its internal weights and biases. More formally, the network parameters w and b get optimized to minimize the cost function given by

$$C(w,b) = \frac{1}{2n} \sum_x ||y(x) - a||^2. \tag{1.2}$$

Here, $y(x)$ denotes the vector of outputs where each element y_j corresponds to the element x_j of the input vector x. w and b denote the collection of all weights and biases in the network, a is the output vector of the network when x is the input and n denotes the total number of training examples. The optimization can be performed using stochastic gradient descent, where the training data is split into randomly picked training inputs $X_1, X_2, ..., X_m$, called mini-batches. Then, the weights and biases can be updated by each mini-batch using the update rules

$$w_k \to w'_k = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k} \tag{1.3}$$

$$b_l \to b'_l = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l}. \tag{1.4}$$

Here, w_k and b_l denote the weights and biases of the neural network, which get updated to the new values of w'_k and b'_l. The step size of the optimization, the so-called learning rate, is given by η, which is a small, positive parameter. The number of mini-batches is given by m and the cost function of the current mini-batch is denoted by C_{X_j}. The weights and biases get updated for each mini-batch in the training set, which causes the network to slowly approximate the function which maps inputs to given outputs. The computation of the gradient for all weights and biases is efficiently done by the so-called backpropagation algorithm [34].

After multiple iterations of the training set, called epochs, and through a large training set with sufficient variety, the approximated function should ideally generalize to

input examples outside of the training set. This means, that for a formerly unknown input example, the network yields an output which matches the output generated by a human.

1.3.2 Convolutional Neural Networks

Besides plain neural networks as discussed in the previous section, there are other network architectures which outperform these plain networks on multiple tasks. One type of such architectures are so-called convolutional neural networks (CNNs).

If the inputs of an artificial neural network are for example pixels in an image, these pixels underlay a spatial structure, meaning that pixels which are next to each other have similar pixel intensities. A fully connected neural network does not make use of this spatial structure, as it connects all pixels to all neurons in the first hidden layer and therefore makes no difference between pixels which are far apart and pixels which are close together. CNNs overcome this issue by using so-called local receptive fields. Each neuron in the first hidden layer is connected to only a small region of the input neurons, which could for example contain 25 pixels in a 5×5 region. By sliding this receptive field over the input neurons, it builds up the first hidden layer consisting of 24×24 neurons. Formally the output $o_{j,k}$ of the j, kth hidden neuron is given by

$$o_{j,k} = g(b + \sum_{l=0}^{4} \sum_{m=0}^{4} w_{l,m} a_{j+l,k+m}). \tag{1.5}$$

The input at position x, y is denoted by $a_{x,y}$, $g(\cdot)$ again denotes an activation function as for example the sigmoid function. The weights $w_{l,m}$ and the bias b build up a 5×5 receptive field, also called kernel. It is important to note that all neurons in the hidden layer use the same kernel and therefore the same weights and bias. This way, the CNN uses spatial structures detected by the local receptive field and makes these structures available for all neurons through using a kernel with fixed weights and bias. If the kernel is for example optimized to recognize vertical lines it may be used for instance in the upper right part of the image as well as in the lower left. The reusability of the kernels also lets CNNs adapt to the translation invariance which most images have.

A CNN uses several kernels, creating multiple layers from one single input image, so-called feature maps. Each feature map, for example 24×24 in size is defined by its own kernel with its unique weights and bias. Therefore one feature map could highlight regions with vertical lines, while another feature map could highlight horizontal lines as

their kernel weights adapted to the respective task.

The shared weights and bias of the kernels greatly reduce the optimization parameters of a CNN and therefore accelerate training. Suppose training a CNN and a neural network with images of 28×28 pixels. If the CNN for example uses 20 kernels with $5 \times 5 = 25$ shared weights and a single bias, a total of $20 \times 26 = 520$ parameters must be optimized during training. If one would instead use a neural network with $28 \times 28 = 784$ input neurons and 30 hidden neurons, it would need $23,550$ parameters and would probably perform worse than the CNN [27].

With these advantages, CNNs have outperformed plain neural networks and other traditional machine learning approaches in the field of computer vision and are the state-of-the-art technique when it comes to images [22][21].

1.3.3 Convolutions for point clouds

Despite the success of CNNs, architectures using the convolutional operation are restricted to ordered and structured data. However, many types of 3D data do not fulfill these requirements. Multiple applications like autonomous driving, medical treatment or robotics benefit from 3D data as it provides more information about the geometries, shapes and scale of the surroundings. With the rapid development of new and more affordable 3D sensors like for example LiDAR, 3D data sets become increasingly available in different formats such as point clouds, meshes and volumetric grids, where point clouds and meshes cannot be processed with architectures based on the traditional convolutional operation.

There exist multiple techniques which leverage the architectures of the structured data realm to deal with all kinds of 3D data [11][13]. With mapping unordered and sparse 3D data onto a grid of voxels, most architectures designed for analysing structured data like images can be applied. However, this approach comes with a large overhead of computation and storage cost due to the sparsity of 3D data. Most of the voxels are empty but must still be processed when using the normal convolution operation. Another method for leveraging the advantages of structured data algorithms for unordered data is the so-called multi-view approach [36]. Here, 3D structures are approximated by taking multiple 2D images from different view points, processing these images with the regular techniques and projecting them back onto the original 3D structure. Although this method outperforms multiple 3D architectures it still comes with additional computing

costs for producing the projections.

Recently, architectures have been developed to deal with unordered 3D data directly. A commonly used format are point clouds, which are unordered structures with irregular density. Point clouds cannot be processed using conventional convolutions as the applications of such operators would result in a dependency on the ordering of the points and in the loss of shape information [23]. This can be illustrated by the point clouds shown in Fig. 1.3. Convolving a kernel of size 2×2 with these point clouds would result in $f_1 = k_1o+k_2r+k_3g+k_4b$, $f_2 = k_1o+k_2r+k_3g+k_4b$ and $f_3 = k_1r+k_2g+k_3b+k_4o$, where f_i stands for the result of the convolution and k_i and o, r, g, b are the kernel elements and point features as shown in Fig. 1.3. With $f_1 = f_2$, the shape information of the respective point clouds got lost, as they are not distinguishable any more. The result of $f_2 \neq f_3$ shows the variance to ordering as the results differ even so the corresponding point clouds are equal.

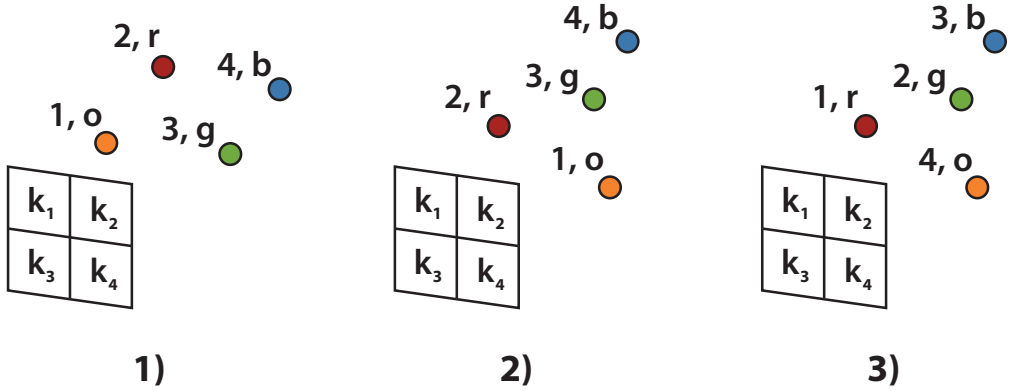

Figure 1.3: The application of a regular convolution to a point cloud results in variance to ordering of the points and in desertion of the cloud shape. The colors o, r, g, and b of the points represent the point features which get multiplied with the kernel elements.

There have been many proposals for architectures which solve these problems, which are all summarized in [11]. Starting with the PointNet architecture from 2016 [31], most proposals used convolution based networks. In this thesis, the ConvPoint architecture was used, which was proposed by Alexandre Boulch in 2019 [6]. This architecture uses a generalization of the discrete convolution operation and replaces it with a formulation which can be directly applied to sparse input point clouds, solving the above mentioned problems and not restricting the input size. Considering a single convolution operation with n kernel elements w_i and n inputs x_j, equation 1.5 can be generalized to

$$o = \sum_{i}^{n} \sum_{j}^{n} w_i a_j 1(k_i, p_j). \tag{1.6}$$

Here, the order of inputs and kernel elements was made explicit and the activation function and bias have been removed for clarity. $1(\cdot, \cdot)$ is the indicator function with $1(a, b) = 1$ if $a = b$ and 0 otherwise. w_i and a_j are the feature values of the respective kernel or input elements and k_i and p_j are the respective positions of these elements. For structured data like images, this definition is well defined and $1(k_i, p_j)$ will always be non-zero. However, with unstructured point clouds, the opposite is the case and the indicator function would be zero most of the time, as it is very unlikely that a point is exactly at the position of a kernel element.

Thus, the ConvPoint architecture replaces the indicator function with a weighting function, which takes the distance of kernel element and points as input and outputs a scalar with which the corresponding kernel point pair should be weighted. Equation 1.6 can then be reformulated to

$$o = \sum_{i}^{n} \sum_{j}^{n} w_i a_j \phi(p_j - k_i) \tag{1.7}$$

where ϕ is the weighting function. A visualization of the ConvPoint architecture is shown in Fig. 1.4.

Intuitively, ϕ would be some kind of Gaussian function, decreasing with the distance of the points. However, in order to avoid any restrictions on the network, the weighting function is also calculated by a simple, additional neural network. This way, the parameters of the weighting function are learned during training and can be optimized as needed. Using this new type of convolution, the kernel elements are given as a point cloud with random positions on for example the unit sphere. The positions and the weights of the kernel elements get optimized during training and the number of kernel elements and points is independent from each other. As the convolution should still be a local operation, it should be applied to some neighborhood of an input point, so the points in equation 1.7 are a subset of the input point cloud.

The described architecture is invariant to the ordering of the input points, solving the first problem when generalizing convolutions to 3D. The architecture can also learn the shapes of the point clouds, as the weighting function will assign different weights to each kernel

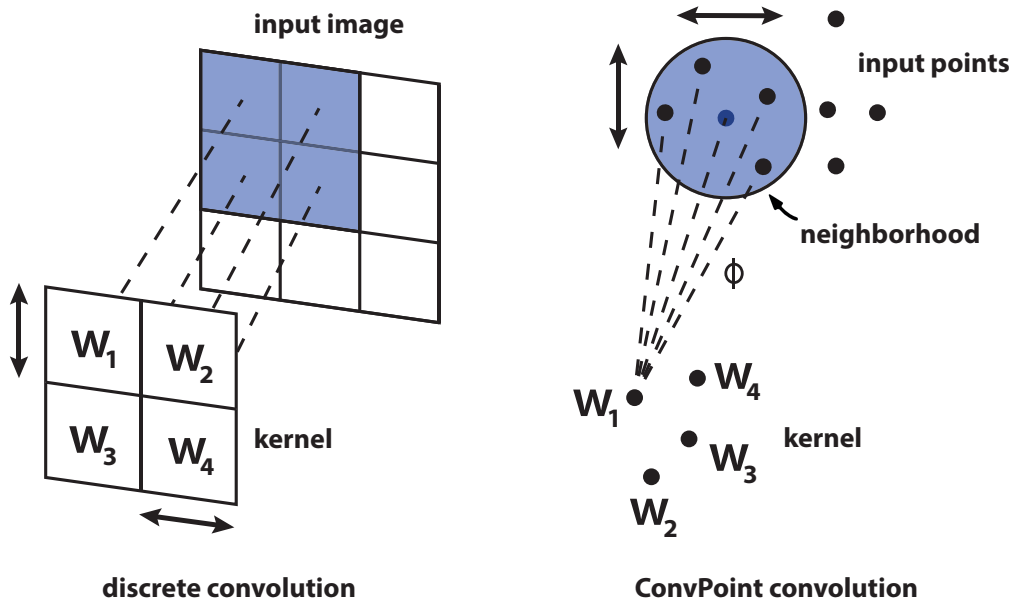

Figure 1.4: Left: Discrete convolution for 2D images. Right: ConvPoint convolution for 3D point clouds. The pixel of the input image and the points of the input point cloud can have additional features (e.g. pixel intensities).

point pair based on their distance. Furthermore it is invariant to global translations as the weighting function takes the relative distance of the points as input.

1.4 Connectomics analysis

1.4.1 Flood Filling Neural Networks

With the increased availability of large EM data sets, the need for automated reconstruction of the neural wiring has become essential. Even with computer-aided data visualization and optimized pipelines, the manual reconstruction of a volume would result in more than 100,000 hours of human labor [15]. The rapid development of CNNs and their variations have improved these reconstructing efforts and CNNs are now an essential part of the analysis pipelines. Starting with 3D EM data sets, like the one from Area X of the zebra finch which was mentioned previously, the first step of the analysis pipeline is the segmentation of neurite structures and their clustering into so-called supervoxels. One state-of-the-art technique are flood-filling networks (FFNs) which are based on CNNs with a recurrent loop [15].

The network is trained on annotated ground truth data of for example 49 × 49 × 25 voxels and consists of two input channels. The first channel takes the actual 3D data as input, while the second channel realizes the recurrent loop and takes the current prediction of the object shape as input, the so-called predicted object map (POM). With this recurrent loop, the network can make use of voxels which were already classified and considers its decisions in the past for further segmentation. With this technique, a mean error-free neurite path length of 1.1 mm could be achieved, which was a performance improvement of an order of magnitude relative to other approaches applied to the same data set. The FFNs create segmented supervoxels (SVs) which get then agglomerated to super-SVs (SSVs) [35]. Each SSV is then corresponding to a single neuron in the data set, an example for such a reconstructed cell can be seen in Fig. 1.5.

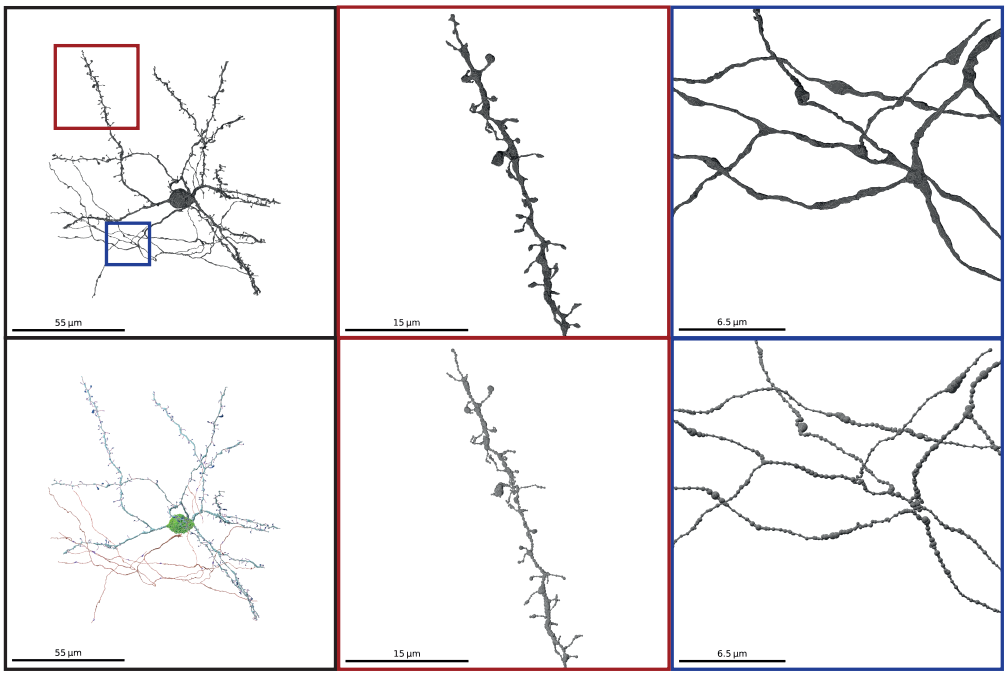

Figure 1.5: A reconstructed cell. It consists of a mesh representing the cell surface and an additional skeleton which represents the coarse structure of the cell. Left: top shows the entire neuron, bottom shows the skeletonized neuron with ground truth annotations (soma: green, dendrite: blue, axon: brown, boutons: purple, terminals: dark red, spine necks: turquoise, spine heads: dark blue). Middle: top shows an enlarged version of a dendrite (red framed region in entire cell), bottom shows the corresponding skeleton. Right: top shows an enlarged version of an axon (blue framed region in entire cell), bottom shows the corresponding skeleton.

1.4.2 Analysis pipeline and SyConn

In order to gain a better understanding of the neural wiring, the reconstructed cells must be further examined. The automated analysis of the neurons consists of essentially four steps: the detection of ultrastructural objects, the removal of glia cells, the classification of neuronal cells and the semantic segmentation of cellular compartments.

In addition to the neuron reconstructions which were generated by the FFNs, cell organelles like mitochondria or vesicle clouds hold valuable information for the prediction of the cellular compartments in the later process. In order to generate a neuronal connectivity matrix the synaptic junctions between different neurons need to be reconstructed as well. Therefore, the first step of the data analysis after the cell reconstruction is the detection of these ultrastructural objects [9]. Data sets acquired with VEM techniques and heavy-metal staining first contain all cell types, including glia cells. Therefore, the second analysis step involves the filtering of neuronal cells. Besides neurons, one main component of the central nervous system are glia cells, which occupy the interneuronal spaces and separate the neurons from blood vessels (blood-brain barrier). Glia cells do not conduct action potentials and do not establish synapses to other cells. Generally, they mainly act as the supporting tissue for neurons and are therefore less interesting for neural circuit analysis [12]. Thus, the second step is the identification and removal of glia cells, as they are not taken into account for the connectomic analysis. The third step of the data evaluation includes the morphology based classification of neuronal cells. As mentioned previously, neurons can take different shapes and can be classified into different cell types like for example excitatory axons (EA), medium spiny neurons (MSN), pallidal-like neurons (GP) or interneurons (INT). As these neuronal types have different properties and functions, their classification is an essential step of neural circuit reconstruction. The fourth step is the identification of cellular compartments and a high-resolution semantic segmentation of neurite surfaces. As described in the section of structural neurobiology, the compartments consist of soma, dendrites and axons. The high-resolution semantic segmentation aims to identify boutons *en passant*, axon terminals and spines. For a more detailed analysis the spines can be further segmented into spine neck and spine head, which allows a classification of the different spine types mentioned earlier.

This whole process from the cell reconstruction to the connectivity matrix and the fully segmented neurons can be unified in a single pipeline. One example for such a pipeline is SyConn, a framework for the automated analysis of EM data from neural tissue, which was first published in 2017 [9]. Since then, the framework has been under active development. In the first step, cellular organelles are detected by 3D CNNs which process

voxelized SBEM image data. The glia detection and removal is done by so-called cellular morphology neural networks (CMNs) [35]. These CMNs are based on CNNs which use the mentioned multi-view approach to analyse multi-channel 2D projections of cell reconstructions. The neuron classification is also done by CMNs. A global, single view of the entire cell would come along with the sacrifice of important details due to reduced resolution. Therefore, many local views from different locations get combined to a global representation of the cell which is then used as input to a CMN. The compartment prediction and dense segmentation of neurite surfaces is also done by CMNs, using varying fields of view (FoVs) to account for the different resolution demands. The last step of the SyConn pipeline consists of the construction of a neuronal connectivity matrix, which is based on the reconstructed and classified contact sites between cells. The results of this connectivity matrix for the j0126 zebra finch data set have been shown to be consistent with the results from previous electrophysiological studies, verifying the results of the SyConn pipeline [9]. The detailed analysis of the connectivity matrix gave insights into biological learning mechanisms [19].

1.5 Problem statement

This thesis is concentrated on step 4 of the data analysis, the identification of cellular compartments such as soma, axons and dendrites. Furthermore, the dense semantic segmentation of neurite surfaces and therefore the identification of finer substructures such as spines (divided in spine head and spine neck), boutons and axon terminals was also tested. Until now the SyConn pipeline uses CMNs with the multi-view approach to tackle these segmentation tasks. However, this technique requires an overhead of computational cost as the views must be generated, stored, processed and mapped back onto the original neuron structure. When processing larger data sets, this computation overhead could be critical and the multi-view approach could therefore not scale well. Another disadvantage of this approach is that an increasing context represented in a single projection makes it more difficult to decode the depth information from the 2D projections due to occlusions.

Recently, it has been shown, that newly developed neural network architectures for the 3D domain can reach the performance of the multi-view approach by processing point clouds directly [6]. Single reconstructed neurons have only sparse coverage of the 3D volume as thin processes span over wide distances. Therefore, 3D-architectures which are based on the voxelization of this volume would have to process many empty voxels which would result in unnecessary computations. However, the generalized version of the convolution operation which was described earlier does not consider voxel, but is able to process point clouds directly without much computational overhead. This thesis aims

to show that it is possible to replace the multi-view approach with point cloud based architectures like the aforementioned ConvPoint network and demonstrates this for the detection of three major functional compartments. Furthermore, this thesis presents a first extension of the semantic segmentation to very fine structures like spines, boutons and terminals.

The starting point for this work were cells which have been reconstructed by FFNs and were provided by Michal Januszewski and Viren Jain from Google Research. They were given as SSVs, which consist of a mesh representing the cellular surface and morphology and a skeleton which is a graph of the cell center line, representing the coarse structure of the cell. The skeleton nodes of a subset of these SSVs were manually annotated into 7 classes (soma, axon, dendrite, bouton, terminal, spine neck, spine head; see Fig. 1.5 and section 2.4) and used as ground truth for training and evaluation of the model. In summary, the overall goal is to use the surface points of cell reconstructions for a semantic segmentation into 3 (axon, dendrite and soma) and 7 (axon, dendrite, soma, bouton, terminal, spine neck and spine head) classes, which ideally matches the human generated annotations.

In order to efficiently work with these data structures, the first step was to develop a framework in which the structures could easily be analysed. The functions of this framework will be discussed in the next chapter.

Chapter 2

Methods

2.1 Problem analysis

The resources to perform the semantic segmentation task are GPUs with a certain memory capacity m in GB. The network which is used with these GPUs has a memory consumption which is dependent on the size of the input point cloud and can be described with $c_b = f(p_{in,b})$ where c_b is the memory consumption in GB, $p_{in,b}$ is the number of input points using a batch size b and $f(\cdot)$ is the consumption function. With equations 1.3 and 1.4 the optimization of the network parameters is done by processing multiple mini-batches. A higher number of batches speeds up the training and inference process and lets trainings converge faster, so the trade-off is between the number of points in a single training example versus the training duration or inference speed. The upper size limit of the input point clouds is implicitly given by $m = f(p_{in,b})$ as sizes above that threshold would lead to memory errors. m depends on the type of graphics card, $f(\cdot)$ depends on the type of network architecture which is used. The mesh of the input cell consists of p_{cell} vertices, the skeleton has p_{skel} nodes, where $p_{cell} >> p_{skel}$. As the skeletons are generated together with the cell reconstructions, they were taken as given and could not be changed. While the skeleton represents the coarse structure of the mesh, its microstructure can vary and take complex forms. An example for such skeleton structures is shown in Fig. 2.1. As the skeleton is an artificial addition to the biologically inferred mesh, it must not influence the result of the segmentation, but only serve as a computational support structure.

Each input cell has a total surface area A_{cell}, which varies with the type and size of the neuron. Dendrites can have a lot of spines which increase the surface area per volume, whereas axons are processes without many protrusions, resulting in a smaller surface area per volume. The total surface area and the number of vertices define the point density $\rho_{cell} = p_{cell}/A_{cell}$ where ρ_{cell} can be varied by using sampling algorithms on the mesh. Here it is assumed that the points are evenly distributed on the surface.

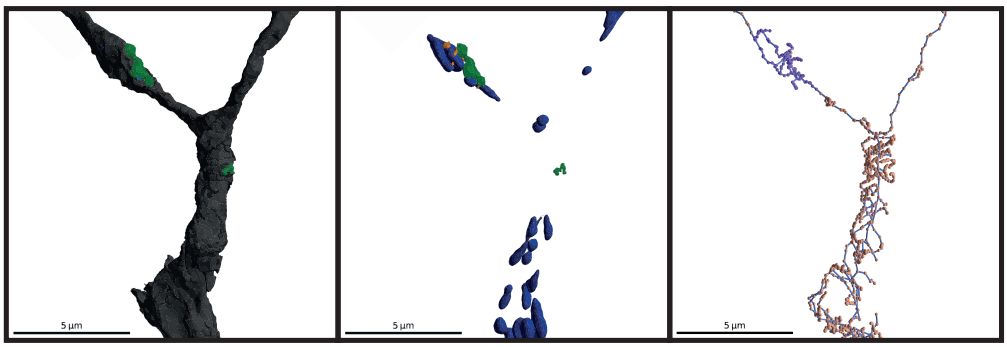

Figure 2.1: From left to right: Mesh structure (synaptic contacts visible in green). Cellular organelles (mitochondria: blue, synaptic vesicles: brown) and synaptic contacts (green). Annotated skeleton with complex, helical microstructure.

The context-dependent, unknown variable ρ_{bio} represents the maximum density below which there are too few points on the surface to reliably identify the cell's morphology. Small protrusions like spines might then degrade into unrecognizable structures and even human experts might not be able to produce a consistent annotation of the given object. Thus, it should always be ensured that $\rho_{cell} > \rho_{bio}$. ρ_{bio} depends on the type of cell and can vary depending on the cell's morphology. Intuitively, ρ_{bio} is relatively high for small, filigree structures like spines, but rather small for pure axon branches without any protrusions.

Furthermore, the cells are enriched with additional meshes which represent the cell organelles and synapses (Fig. 2.1). Besides organelles like mitochondria and synaptic vesicles there are also meshes of segmented synaptic clefts. These additional structures can be exploited during semantic segmentation as their appearance can be a strong indicator for the type of a neuronal compartment. Through preprocessing, it can be ensured that these meshes share the cell's point density ρ_{cell}.

2.2 MorphX

This chapter describes MorphX (short for morphology exploration), the framework which was developed for handling the neuron data in the various processing and analysis steps. First, it provides methods to ensure the uniformity of the data by different sampling techniques on the surface of the given meshes. Second, it contains classes with which algorithms can be applied to point clouds with and without additional skeleton or mesh information. Third, MorphX provides different methods for splitting a given mesh or point cloud into multiple parts, following certain chunking criteria. Fourth, it provides

classes for feeding the generated chunks into a neural network or to apply a pretrained model to the data for inference. Inference differs from training by not requiring the back-propagation step, as the model does not need to be trained but only needs to generate predictions for the given inputs. The code is available at: https://github.com/StructuralNeurobiologyLab/MorphX.

2.2.1 MorphX classes

While there are a few available frameworks for dealing with point clouds or 3D data [20][32], most of these libraries are under active development and still in their infancy. MorphX was created as a rather general framework which might be used in other contexts outside of Connectomics in the future. It contains a small set of classes which implement specific attributes and functionalities with the PointCloud and the CloudEnsemble as base classes. The PointCloud class represents objects which consist of points with labels and features in 3D space and provides different transformations and storing and evaluation methods for point-wise predictions. The HybridCloud class is derived from the PointCloud and represents objects with an additional skeleton with nodes, edges, and possible node labels. Furthermore it provides the mapping of vertices to nodes and has multiple types of graph algorithms which can be applied to the skeleton. The HybridMesh class is derived from the HybridCloud and has additional mesh components like faces and the corresponding face to node mapping which is used in the preprocessing methods. A CloudEnsemble object consists of a HybridCloud object and multiple PointCloud objects and is used to store and organize the actual cell and all the corresponding cell organelles.

2.2.2 Preprocessing

The meshes of single SVs can contain boundary artefacts due to chunk-wise processing of the FFN segmentation and the merging of different SVs to a single SSV. To reduce bias introduced by these systematic errors, a uniform surface representation must be ensured.

In order to remove such artefacts, the initial reconstructed cells are preprocessed with a sampling algorithm which ensures the uniformity of the surface points and therefore ensures an accurate representation of the cell's morphology. One possible sampling approach which was taken in this thesis is a mesh based Poisson disk sampling for which an already existing method from the point-cloud-utils library was used [37]. Poisson disk sampling places the samples at random points, but ensures a minimum distance r between all points. Poisson disk sampling is mostly used in graphics applications like texturing or

point-based rendering and can be efficiently done achieving around 180,000 samples computed per second [7]. With this method, the point artefacts of given cells were eliminated and an uniform point representation of the cell surfaces was ensured.

2.2.3 Splitting mechanisms

Depending on the network architecture and the type of GPUs, there exists an upper limit $p_{in,b}$ for the size of point clouds which can be processed at once. Usually, the total area A_{cell} of a cell is so large, that it cannot be processed at once. Doing so would result in an unsatisfying density $\rho_{chunk} = p_{in,1}/A_{cell} << \rho_{bio}$, even if the largest point cloud size with a batch size of $b = 1$ is used. Thus, large cells must be divided into multiple parts which are processed individually. This chunking ensured sufficiently high resolution in areas with high information content, such as spines.

The chunking mechanisms which were implemented for this thesis were based on the cell's skeleton graph, as the processing of the skeleton is less computationally expensive than operations on the mesh. Starting with all nodes and the corresponding edges between them, a random node is drawn. This base node is then used as the extraction point for a smaller subgraph. The vertices of the mesh get mapped to the nodes of the skeleton according to the Voronoi cells of the nodes. Given the local context of multiple nodes, the mapping information can be used to extract the corresponding mesh chunk, which can then be used as a training or inference example. The nodes contained in the local context get removed from the total amount of nodes and the process starts again by choosing a random base node from the remaining nodes. This way, it is guaranteed to cover the total cell and that the chunks are overlapping such that each vertex gets multiple predictions which can then be combined by a simple majority vote.

A local context which was extracted by a simple radius threshold around the base node would result in a varying density. If for example all nodes within a certain radius of the base node would be chosen for the local context, the context is independent from the actual surface area included in this radius. As the number of points p_{chunk} in that sample could be higher than the maximum cloud size $p_{in,b}$, the maximum number of points must be sampled from p_{chunk}. For regions with a high surface area per volume, this would result in a density $\rho_{chunk} = p_{in,b}/A_{chunk} < \rho_{bio}$ and therefore in a possible degradation of important substructures. To avoid this, the local context can be generated dependent on the currently selected surface area. Starting from the base node, a breadth first search (BFS) is performed until a certain threshold of surface area is reached. Assuming that the vertices are uniformly distributed on the surface, the surface area correlates with the number of vertices included in the local context. Ideally, the area threshold is expressed

by $A_{chunk} = p_{in,b}/\rho_{bio}$, ensuring a uniform density while always processing the maximum number of points. Thus, the BFS is performed until the number of vertices corresponds to A_{chunk} and the resulting vertices build up the training example which is then fed into the network. Using this method, the context size adjusts itself to the surface area per volume in a given area. In cell parts which contain many protrusions like spines or which have dense structures like the soma, the context size will be smaller than in areas with only few protrusions like for example axons. All chunks are represented by point clouds with the same density ρ_{bio}. Both splitting mechanisms, the simple context-based mechanism and the density-based mechanism were tested and used for later processing steps.

Fig. 2.2 shows chunks which were generated using density-based splitting. As described above, it can be seen that the context of the chunks is rather small for regions with high surface area per volume. Therefore, regions around the soma or around complex structures like multiple boutons have a rather small context size, while regions which only include thin axons and small boutons have a larger context. The point density is approximately the same for all regions.

Fig. 2.3 shows chunks extracted by the context-based splitting approach. One possible problem of this context based splitting is that the context is independent from the actual surface area. Around the soma, the surface area is relatively high, but the number of sample points is the same as in all other regions. Depending on the number of sampled points, this leads to the degradation of structures like it can be seen on the axons and dendrites on the right-hand side of Fig. 2.3. The point cloud visualizations were generated with Open3D [39].

With the Voronoi based mapping of vertices to nodes, it can happen that the extracted vertices depend on the microstructure of the skeleton. As shown in Fig. 2.1, the skeleton can contain helix structures, which would cause the vertices to be mapped to the nodes on the helix, but not to the underlying nodes in the center of the helix. This would result in missing vertices in the resulting chunk, obscuring the biological morphology because of the artificial structure of the skeleton. One solution to this problem would be to implement a better skeleton generation algorithm, which was out of scope for this thesis. Instead, a workaround has been implemented, where the BFS extracts all nodes within a certain radius r of the current node in each step. These nodes and their corresponding vertices are then added to the local context in the order of their distance to the current node. The radius r was determined as a local approximation of the cell extension, which bypassed the microstructure of the skeleton and ensured a coherent point cloud.

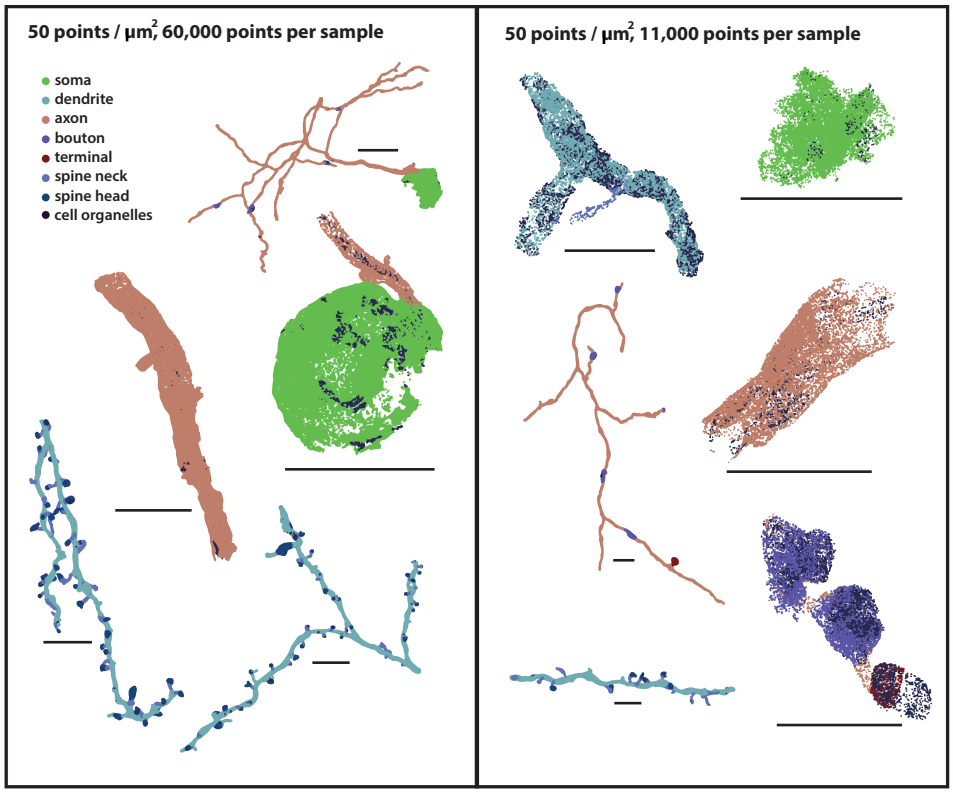

Figure 2.2: Density-based splitting. All chunks have a point density of approximately 50 points per µm^2. On the left the number of sample points was set to 60,000, on the right it was reduced to 11,000. Regions with a large surface area per volume have a small context, while regions with a small area have a large context. On the left, the scale bars have a length of 10 µm, on the right they have a length of 5 µm.

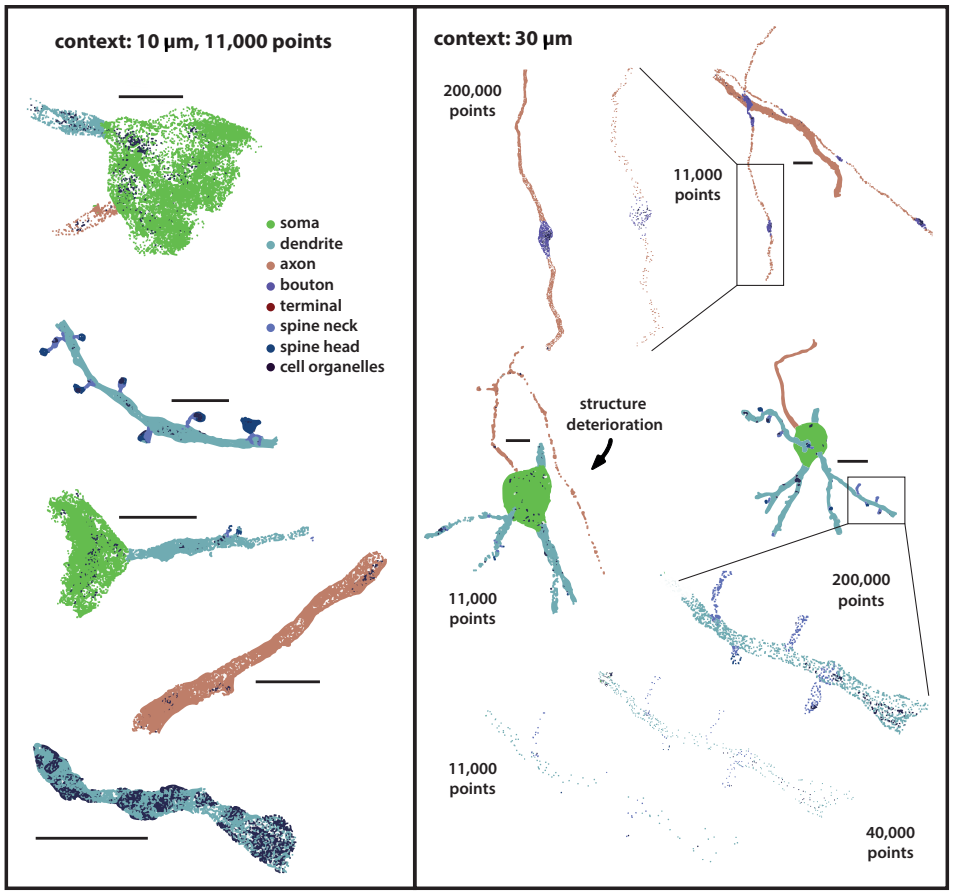

Figure 2.3: Context-based splitting. On the left the chunks have a context of 10 µm, on the right the context was set to 30 µm. On the left side the number of sample points was set to 11,000, on the right side the point number was varied. The context-based splitting can result in deterioration of structures in areas where there is a large surface area per volume. The scale bars have a length of 5 µm in both parts of the figure.

2.2.4 Network architectures

The neural network architecture used in this thesis was taken from [6] and is based on the ConvPoint convolution. The network architecture used for all of the point-wise compartment classification tasks was designed to perform semantic segmentation of the input point cloud. As the ConvPoint convolution is simply a generalization of the discrete convolution, the network architecture of a ConvPoint based NN is quite similar to previous segmentation network architectures as for example the U-Net [33]. It consists of an encoder which reduces the cardinality of the input point cloud while enriching it with more feature maps. The second part of the structure, the decoder, uses this downsampled and feature-enriched point cloud to upsample the point cloud back to the original size and then provides the resulting points as input to a fully-connected linear layer which produces the final output according to the number of classes. Both, encoder and decoder consist of multiple convolutional layers, where the number and attributes of the layers in both parts are symmetric. The layers of the decoder get the concatenated feature maps from the previous decoder layer and the corresponding encoder layer as input and use the same points for upsampling as the encoder used for downsampling.

The specifications of the individual layers can be seen in Table 2.1. Each layer has 4 parameters. The output channel number C is the number of kernels and therefore corresponds with the number of output feature maps. Each layer can change the number of points, so another parameter is the size of the output point cloud Q. As shown in Fig. 1.4, the neighborhood size k corresponds to the section of the image which was considered in the discrete convolution. With the ConvPoint convolution, the number of kernel elements N is independent from the neighborhood size and was set to 16 for all kernels in all layers.

Each layer uses the ReLU activation function $ReLU(x) = max(x, 0)$. Furthermore, each layer uses Batch Normalization [14] which normalizes the layer inputs by evaluating the statistics of each mini-batch. This avoids lower training speeds due to parameter changes of previous layers to which each layer would have to adjust. Additionally, Batch Normalization also acts as a regularizer and can be used against overfitting of the data. The neighborhood calculations are done using the NanoFLANN framework [5]. The network was implemented in Pytorch [29]. All kernel elements of the ConvPoint filters were randomly drawn from the unit sphere.

2.2.5 Hardware and training parameters

All processing steps and trainings were performed on the WHOLEBRAIN cluster of the Max Planck Institute for Neurobiology. The cluster is maintained by Christian

layer	output channels C	output points Q	neighborhood size k
0 conv	64	input size	16
1 conv	64	2048	16
2 conv	64	1024	16
3 conv	64	256	16
4 conv	128	64	8
5 conv	128	16	8
6 deconv	128	8	4
7 deconv	128	16	4
8 deconv	128	64	4
9 deconv	64	256	4
10 deconv	64	1024	4
11 deconv	64	2048	8
12 deconv	64	input size	8
13 linear			

Table 2.1: Layer specifications of the ConvPoint based architecture which was used for the segmentation tasks.

Guggenberger and the team from the Max Planck Computing & Data Facility. It consists of 18 nodes with 20 cores (Intel Xeon CPU E5-2660 v3 @ 2.60GHz) each. Each node has 2 NVIDIA QUADRO RTX 5000 with 16 GB GPU memory and 256 GB of RAM.

The ConvPoint architecture was added to the Pytorch-based elektronn3 library [1] which was developed for the semantic segmentation of volumetric biomedial image data. elektronn3 was then used as a framework for running trainings and for logging the trainings and validations via tensorboard [3].

The trainings mainly used the Adam algorithm as an optimizer [16] and the StepLR learning rate scheduler [29] with an initial learning rate of 0.001, a step size of 1,000 and a learning rate decay or gamma of 0.95. Stochastic gradient descent was also tested for optimization together with a cosine annealing scheduler with warm restarts. Training durations until full convergence of the models varied between 2 and 5 days.

All trainings used random rotations between $-180°$ and $180°$. The points of the samples were centered around their arithmetic mean. For context-based splitting, the points were normalized by the context size, for density-based splitting, all samples were normalized by a fixed factor of 50 or 100 µm.

2.2.6 MorphX pipeline

Fig. 2.4 illustrates the major steps when processing cells with the MorphX infrastructure. First, the cells get preprocessed and converted into the MorphX data standard, pickle files which contain the attributes of the different MorphX objects. For training purposes, the data set was split into 2 parts: ground truth for training and ground truth for evaluating the trained models.

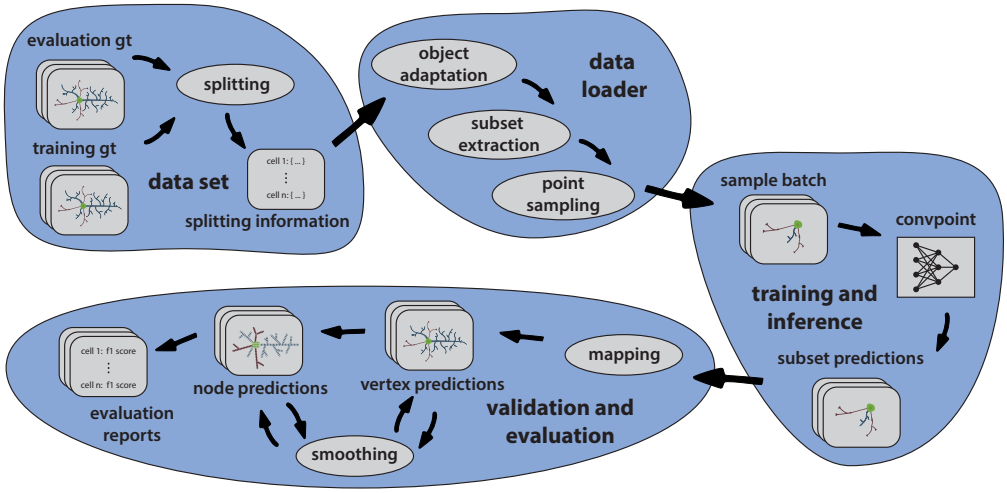

Figure 2.4: MorphX processing pipeline consisting of data set preparation, data loading, training or inference and validation and evaluation. Rounded rectangles indicate files, ellipses represent algorithms. The ConvPoint architecture is represented by the rectangle.

In the first step of the pipeline, the splitting mechanisms described in section 2.2.3 generate splitting information for the given data set. For each cell, the results of the splitting, multiple cell samples, get saved in the form of skeleton nodes with which the chunks can later be generated again. The data loader prepares the data in a way, such that it is presentable to the network architecture. In inference mode, this preparation must comply with the trained model, e.g. sample size and point number per sample should be the same as the model has seen during training. In the first step of the data loader the reconstructed cells get enriched with cell organelles and other structures like reconstructed synapses. For these objects, their surface vertices are added and a one-hot encoding is used to distinguish different types in the feature domain. Thus, all points of each cell part get their own feature vector, e.g. $[1, 0, 0, 0, 0]$ for the points of the cell surface, $[0, 1, 0, 0, 0]$ for the points of the cell surface with myelin, $[0, 0, 1, 0, 0]$ for points corresponding to mitochondria, and so on. The second step of the data loader, the chunk extraction, then uses the previously generated splitting information to produce

individual cell chunks according to the splitting mechanism in use. These cell chunks are point clouds with similar surface area per volume or context size, but the number of points in each chunk can still vary. However, in order to present the chunks to the network in form of batches, the number of points must be fixed. Thus, the last step of the data loader performs a sampling in which a fixed number of random points is drawn from the original point clouds of the cell chunks. The resulting point cloud samples then get stacked and build up a sample batch which gets fed into the network. During training, this sample batch also includes ground truth information which can be used for loss calculation and optimization. The pipeline in training mode ends here and results in a trained model which gets saved for later use.

In evaluation or inference mode, the network architecture produces batches of chunk predictions. These predictions are then mapped back to their original location in the respective cell. This results in vertex-level predictions where each vertex can also have multiple predictions if it is part of multiple cell chunks. Multiple predictions are evaluated by performing a majority vote on the predictions of each vertex. Due to the random sampling on each chunk, not all vertices have a prediction. Depending on the overlap and the number of points drawn from each chunk, a certain coverage of the total number of vertices in the cell is reached. By processing each cell multiple times this coverage is increased until it satisfies the requirements for the current task, which depend on the needed resolution.

In order to get a lightweight version of the vertex predictions and to get a full prediction coverage of the cell surface, the vertex predictions are mapped onto the skeleton nodes by the aforementioned Voronoi based mapping, where each vertex gets assigned to its nearest node. For each node, a majority vote on the predictions of the corresponding vertices is performed and the result is set as the predicted node label.

In inference mode, the pipeline yields an annotated skeleton which can then be saved for later, biological analysis. In evaluation mode, these annotated skeletons and the vertex level predictions were used to generate performance reports which indicate the performance of the trained model on different classes and metrics.

2.3 Evaluation metrics

In compliance with [9] and [35], the F_1-score was used to evaluate the model performance. The F_1-score, also known as the Dice coefficient is defined as the harmonic mean of precision and recall:

$$F_1 = 2\frac{P \cdot R}{P + R} \tag{2.1}$$

P is the precision defined by

$$P = \frac{tp}{tp + fp} \tag{2.2}$$

where tp is the number of true positives and fp is the number of false positives. R is the recall defined by:

$$R = \frac{tp}{tp + fn} \tag{2.3}$$

where fn is the number of false negatives. Besides the F_1-score, the accuracy is used as another performance metric which is defined by

$$a(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{n_{samples}} 1(\hat{y}_i = y_i) \tag{2.4}$$

The metrics were calculated by the respective functions in the scikit-learn python library [30].

The evaluation of the models was done on node basis, where all vertex predictions have been mapped and merged into their corresponding skeleton node by majority vote. For all evaluations, each cell of the evaluation set was processed 5 times.

2.4 Ground truth

The ground truth for training and evaluation was built on previous ground truth data used in [35]. Additionally, multiple cells have been annotated by Julian Hendricks. Further refinements were discussed with Philipp Schubert and carried out by myself. The annotation was done on skeleton level using the KNOSSOS annotation software [2]. The skeleton annotations were then mapped onto the mesh by the Voronoi mapping method. Altogether, the ground truth included 25 annotated cells, which were split into two data sets, one for training and one for evaluation.

2.4.1 Training set

Fig. 2.5 gives an overview of the cells used for training. The total training set has a myelin coverage of 7.6%. It consists of 33.8% dendrites, 25.1% axons, 26.7% soma, 6.3% boutons, 2.1% terminals, 2.9% spine necks and 3.0% spine heads. The total cell surface area sums to 70,802 µm^2. In total, there are 4,374 mitochondria with a combined surface area of 10,187 µm^2, 6,068 synaptic junctions with 3,644 µm^2 and 856 objects which represent vesicle clouds with an area of 1,286 µm^2. Detailed specifications and images of all cells can be found in Appendix A.

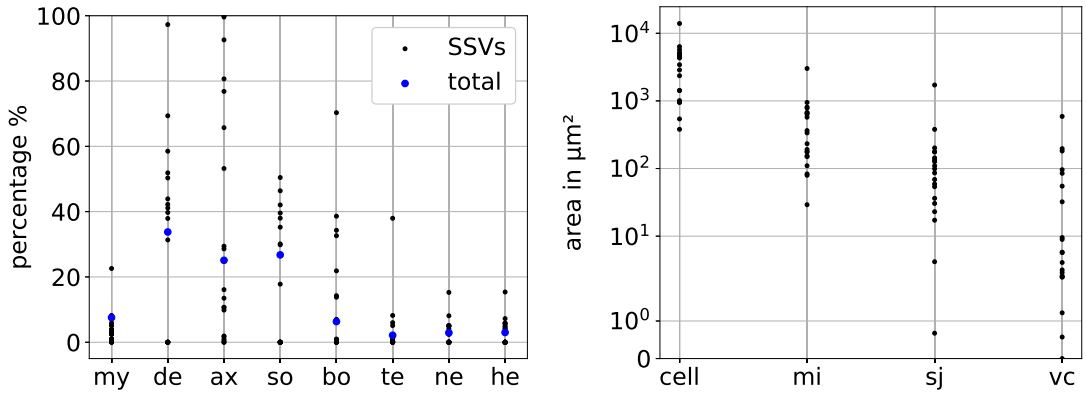

Figure 2.5: Left: Overview of the occurence of different cell parts in the training set. The percentage of myelin (my) is independent from the other classes, respective to the total cell. The percentages of the other classes add up to 100% (dendrite: de, axon: ax, soma: so, bouton: bo, terminal: te, spine neck: ne, spine head: he, total cell: cell, mitochondria: mi, synaptic junctions: sj, vesicle clouds: vc). Right: Overview of the surface areas of different subcellular structures in the training set.

2.4.2 Evaluation set

Fig. 2.6 shows the relative composition of the cells on the left and the size of the surface area of different objects on the right. The total myelin coverage in the evaluation set accounts to 7.7%. The evaluation cells have 24.6% dendrites, 37.9% axons, 17.2% soma, 12.8% boutons, 2.3% terminals, 2.5% spine necks and 2.7% spine heads. The total cell surface area is 15,463 µm^2 with 1,154 mitochondria (2,263 µm^2), 1,129 synaptic junctions (888 µm^2) and 304 vesicle clouds (629 µm^2) (appendix A).

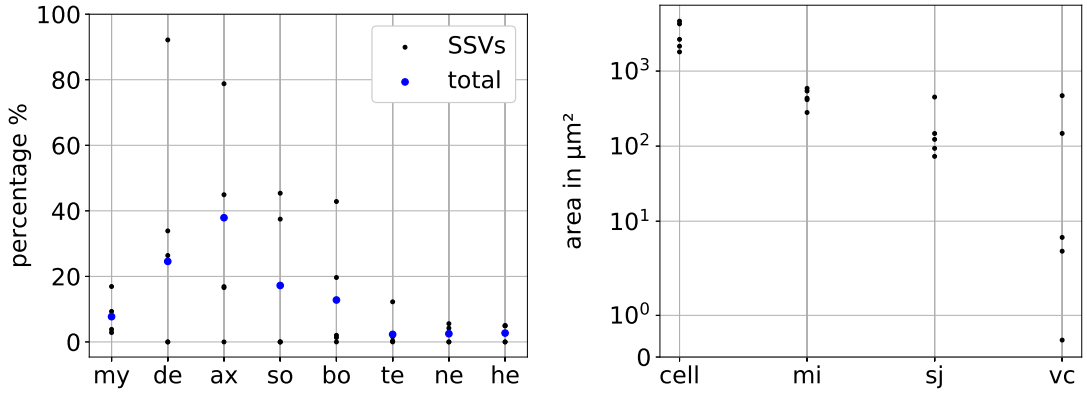

Figure 2.6: Left: Overview of the occurence of different cell parts in the evaluation set. The percentage of myelin (same abbreviations as in Fig. 2.5) is independent from the other classes, respective to the total cell, the percentages of the other classes add up to 100%. Right: Overview of the surface areas of different subcellular structures in the evaluation set.

Chapter 3

Results and Discussion

3.1 Timing evaluation

Due to the size of large connectomic data sets, a manual analysis and annotation is not reasonable. Even an automated data analysis pipeline like the one presented with SyConn must be fast in order to scale well with the massive amounts of data that are required to reconstruct large neural circuits. This section provides an estimation of the current processing times for the semantic segmentation of reconstructed cells with the MorphX pipeline.

3.1.1 Timing of cell splitting

As shown in Fig. 2.4, the first step is to split all cells in the data set using one of the splitting mechanisms provided by MorphX. As the splitting mechanisms are purely based on the skeleton, the computation times are also dependent on the skeleton structure. As mentioned before, the current skeletons have rather complex microstructures which slow down the splitting by requiring costly calculations in order to generate reasonable samples. However, this will be improved in the future by replacing the skeletons with cleaner and simpler versions which will not require the additional computations. Fig. 3.1 shows the dependency between the time needed for splitting and the number of nodes in these new skeletons. For the 25 cells included in the training and evaluation set, there is a linear dependency between the splitting time and the skeleton size. For larger skeletons, more subgraphs must be generated to produce the chunks, which increases the computation time. The context based splitting mechanism results in similar processing times.

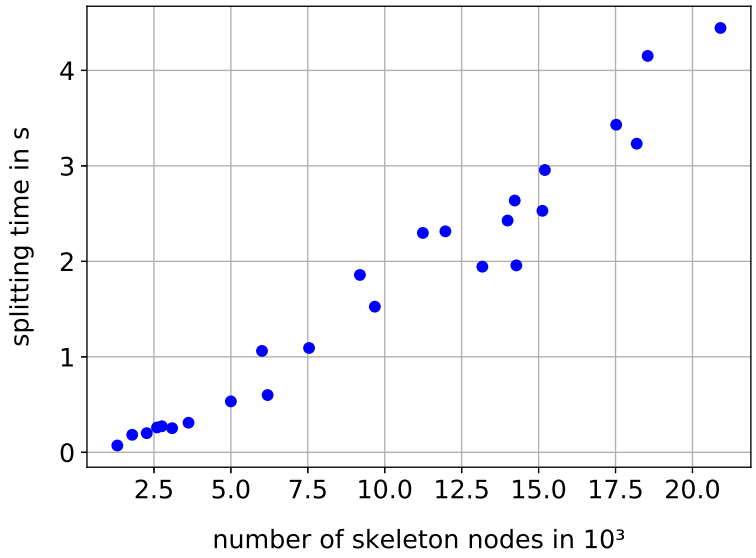

Figure 3.1: Timing of data splitting. Each data point represents one cell from the training or evaluation set. The splitting was done using the density-based splitting mechanism with a density of 50 points per μm^2 and 32,768 sample points.

3.1.2 Timing of the ConvPoint model

Fig. 3.2 shows a timing estimate for the pure network processing step in inference mode, where no backpropagation is needed. Thus, the model can process more batches of samples at the same time than during training. To get a better understanding of the dependency between processing time and number of points included in the input, the network was timed for samples of different sizes. Fig. 3.2 shows the results of this timing. For each sample size, the maximum number of batches (bounded by the GPU memory) was calculated and used for processing. During training, the maximum number of points for one sample was around 120,000 points which is limited by the GPU memory of the graphics card in use.

A good estimate for the size of a large cell as represented by a point cloud is around 10^6 points per cell. Processing an entire cell therefore results in a processing time of about 1.5 seconds when using a sample size of 80,000 points (see Fig. 3.2). Note that this does not include the loading time for new batches.

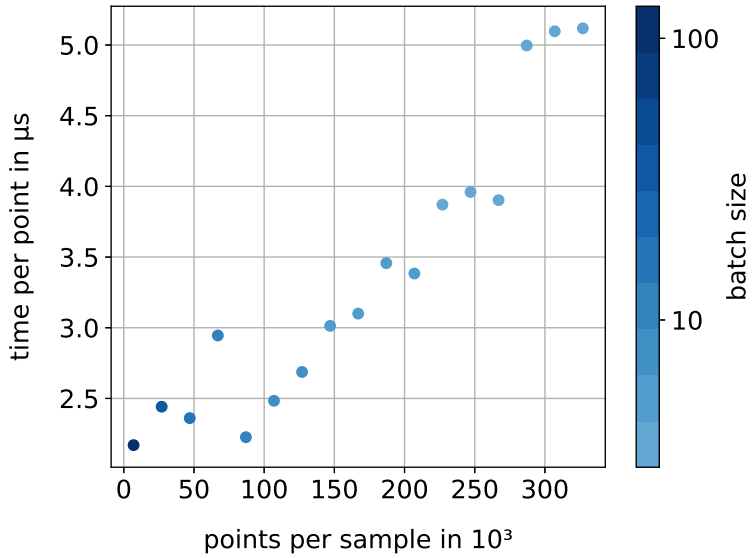

Figure 3.2: Timing of the ConvPoint model. Data points show processing times for random point samples of different sizes. The processing times are given in µs per point. All samples were processed with the maximum possible batch size for the graphics card in use.

3.1.3 Timing of the validation pipeline

This section provides an estimate of the mean processing time per cell, depending on the coverage of the cell. The coverage indicates the fraction of points with a predicted label over the total number of points in the cell.

Fig. 3.3 shows timing results for the full inference of the 25 cells included in the training and evaluation set (appendix A), processed with different coverages. The timing was done using previously generated splitting information with density-based splitting (50 points per μm^2 and 28,000 sample points). To achieve the different coverages, the cells were processed multiple times, each time randomly selecting different points in the respective chunks.

In order to be able to vary the point density of the chunks during density-based splitting, the point cloud representations of the cells used in Fig. 3.3 were heavily oversampled. Point cloud representations which are solely based on the vertices of the cell mesh would have around 60% less vertices. Therefore, a good estimate of the actually needed processing time is given by the data points corresponding to a coverage of 8%. For example, a cell in Fig. 3.3 with 10 million points has only 4 million vertices in its mesh structure

when no oversampling was used. With oversampling, a coverage of 8% results in 800,000 points with predictions. Without oversampling and using the vertices of the original mesh, these 800,000 points account to a 20% coverage of the 4 million vertices in the cell mesh. Therefore, with the current state of the MorphX pipeline it is possible to generate reasonable node-based predictions for most cells in under 25 seconds.

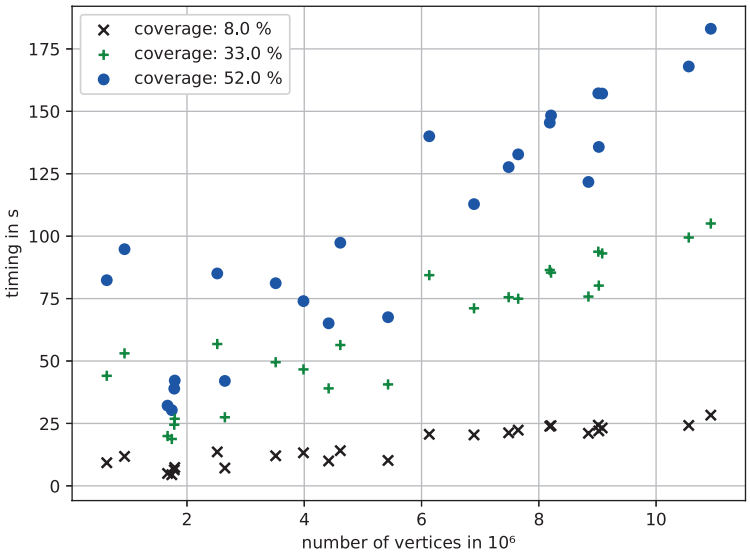

Figure 3.3: Timing of full inference pipeline for different coverages. The data points represent the 25 reconstructed cells of different sizes included in the training and evaluation sets (described in appendix A). The timing includes the data loader steps, the processing of the sample batches by the model and the mapping of the predicted samples onto the original cell.

3.2 Effects of context size and number of sample points

In [9] and [35], it was reported, that the size of the local contexts is an important parameter which heavily influences the performance of the models. For the multi-view approach, a large context seemed to be necessary to be able to determine the type of branch based on the information of the surroundings. Reducing the resolution of the images used for the multi-view approach in [35] had only minor effects on the models.

For the ConvPoint approach presented in this thesis, the resolution relates to the point density of the training chunks, while the context is determined by the splitting method in use. Multiple trainings were performed to analyse the effects of these hyperparameters for

the ConvPoint model. First, models were trained to perform the 3-class task, including the three major cell compartments soma, dendrite and axon. In a second experiment, the models were trained to discriminate 7 classes, now including also finer compartments such as boutons, terminals and spines divided into spine neck and spine head. For all trainings, the samples were enriched with cell organelles, excluding myelin. The effects of cell organelles and myelin can be found in section 3.4. The evaluation set is described in section 2.4.2 and in appendix A. All performance results reported in this section are F_1-scores on the total evaluation set, calculated by combining the node-level predictions of all 5 evaluation cells and then averaging over the class-wise F_1-scores.

3.2.1 Models for dendrite, axon, soma prediction

Fig. 3.4 shows two different evaluations of the same set of trainings with context-based splitting at different times. The evaluations were done on skeleton node-level, so vertex-wise predictions of the mesh have been mapped to skeleton nodes by performing a majority vote on the corresponding vertices of each node. The first evaluation was done before convergence of the models, the second, final evaluation was done when all trainings were converged. As shown in Fig. 3.4, the performance of some models decreased over time. This can be explained by overfitting to the training set, where the function which was approximated by the model is not generalizable to examples outside of the training set. This is normally prevented by validating all models during training on an additional ground truth set and stopping the trainings as soon as the performance on the validation set decreases. While the variety and size of the training data is crucial for the use of artificial neural networks, the generation of new ground truth is time consuming and expensive. The currently available ground truth with 25 annotated cells which was used for the experiments presented here is rather small. Therefore, the splitting of the available ground truth and the generation of an additional validation set could not be done.

The resulting performance of the models using different hyperparameters do not show a linear dependency between predictive performance and context size or number of points. Given the performance of the models trained with 28,000 points in Fig. 3.4 on the left, it is observed that the context size does not have strong effects on the model performance. Except for the model trained with 60,000 points, it is further observed that a higher resolution, meaning a higher number of sample points, leads to better performance. The training with 60,000 points might still have surpassed the one with 40,000 points at the context size of 30 µm, supporting this hypothesis. However, due to possible overfitting and variance of the final model performance, the measurements on the right-hand side of Fig. 3.4 show different results, where the performance of the models with 80,000 and

60,000 points has fallen below the scores of models with less points.

The model with the highest performance was found at the intermediate evaluation (F_1-score of 0.97 with a context of 40 µm and a sample number of 80,000 points) and is evaluated in more detail in section 3.3.1.

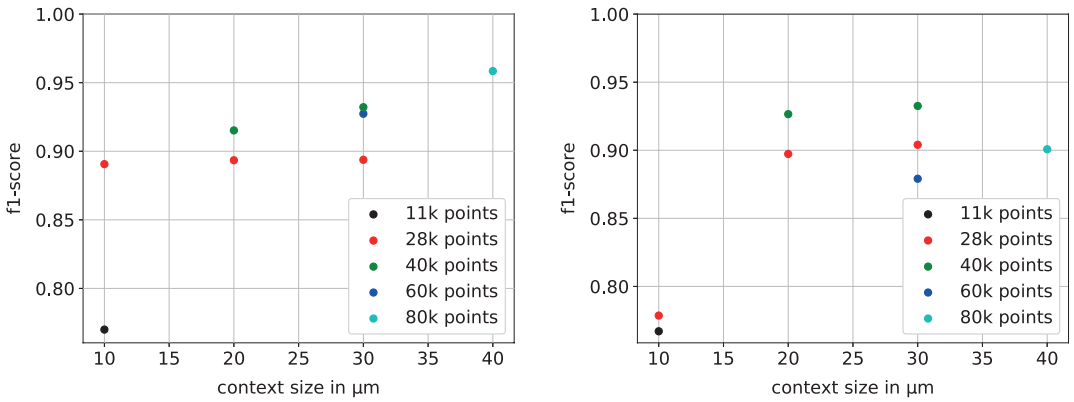

Figure 3.4: Skeleton node-level evaluation results from the parameter search with context-based splitting for the 3-class task. Each data point represents one training with indicated parameters. Left: An intermediate evaluation of the trainings. Right: The final evaluation after all trainings have converged.

In addition to the context-based splitting, multiple models have also been trained using the density-based splitting. Again, Fig. 3.5 shows skeleton node-level evaluations of the same set of trainings at different times. Here, most measurements are stable or improve over time, only the training using a density of 200 points per µm^2 had a worse performance after convergence than at the intermediate evaluation, which again might be caused by overfitting.

The data in Fig. 3.5 shows no linear dependency between the performance and the density or number of sample points. As shown in 2.2, the density-based splitting generates samples, where the context size is adapted to the surface area of the samples. The high performance of the density-based models show that the ConvPoint model is able to adjust itself to these variety of context sizes. However, the comparison between models using density-based splitting and the ones using context-based splitting as shown in Fig. 3.4, indicates that a fixed point density per sample is not necessary. Models with a density-based splitting method show lower scores than the context-based ones. As shown in Fig. 2.3, the context-based splitting can result in the degradation of

filigree structures which was suspected to have a negative effect on the performance and which was one of the reasons for the development of the second, density-based splitting method. Considering the results presented in Fig. 3.4 and Fig. 3.5, the models seem to be able to adjust surprisingly well to the varying point densities of the chunks.

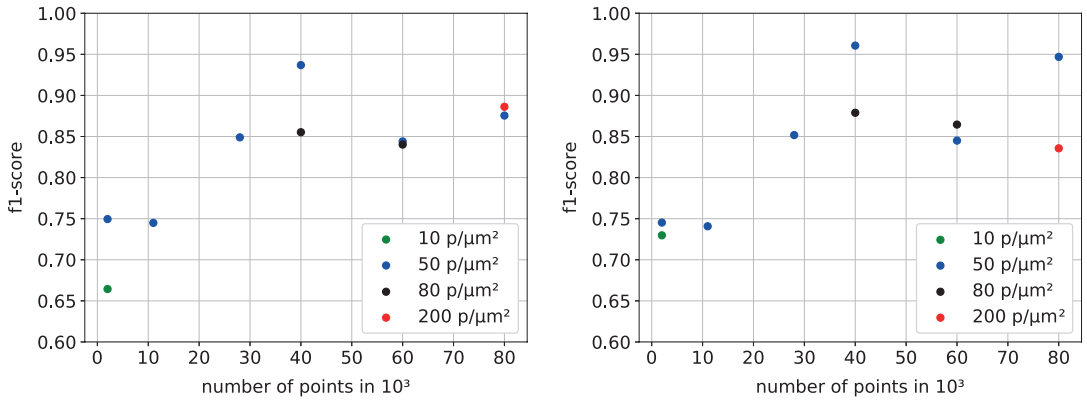

Figure 3.5: Skeleton node-level evaluation results from the parameter search with density-based splitting for the 3-class task. Left: An intermediate evaluation of the trainings. Right: The final evaluation after all trainings have converged.

3.2.2 Models for spine, bouton and terminal prediction

Fig. 3.6 shows the performance of different models (context-based on the left-hand side, density-based on the right-hand side) trained for the prediction of all 7 classes (soma, axon, dendrite, boutons, terminals, spine necks and spine heads). The results shown are the highest scores achieved during an intermediate evaluation. As for the 3-class task, the results of the 7-class task show no clear correlation between the performance and the number of points per sample. For the 7-class task, it is observed that the number of points has almost no effect on the performance. For the context-based models, a larger context has slightly positive effect on the evaluation scores. For the density-based models the present data does not allow any direct conclusions about the best choice of point density. The model with the highest F_1-performance found for the 7-class task, was one with context-based splitting using a context size of 40 µm and 100,000 points. The detailed evaluation of this model is found in section 3.3.2.

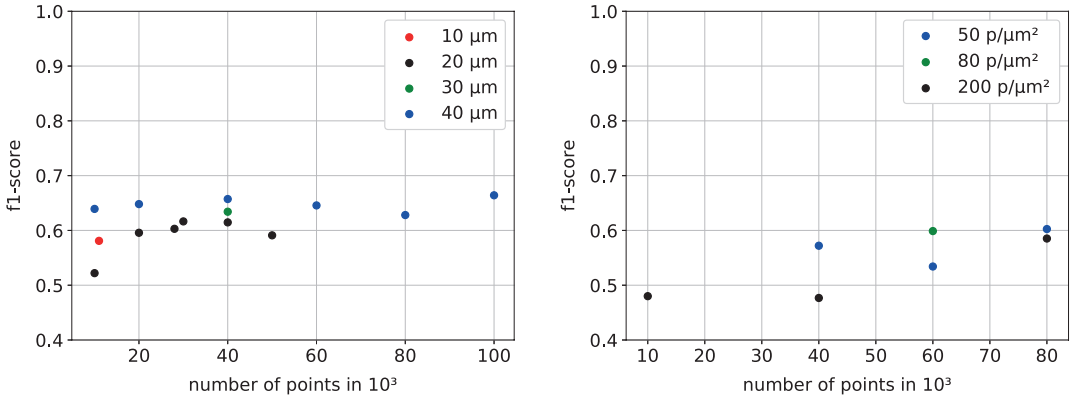

Figure 3.6: Skeleton node-level evaluation results of models trained for the 7-class task. Left: models with context-based splitting. Right: models with density-based splitting. Each data point indicates a training with different parameters. Shown are the highest scores found during the intermediate evaluation.

3.3 Detailed evaluation of the best performing models

3.3.1 Dendrite, axon, soma predictions

The best 3-class model for the axon, dendrite and soma prediction found during the parameter searches described in section 3.2 was one with context-based splitting using a radius of 40 µm and 80,000 points. It was evaluated by processing each cell in the evaluation set 5 times, resulting in a total vertex prediction coverage of 26%. An evaluation with a higher coverage of 40% did not show significant improvements but resulted in almost the same performance results.

Table 3.1 shows the evaluation on skeleton node-level. The metrics were calculated on the combined set of all cell skeletons, meaning, that the node-level predictions of all cells have been merged and then evaluated by the metric. The 'macro avg' represents the unweighted average of the metrics of all classes. The 'weighted avg' score was computed by the support weighted average of all classes.

The evaluation shows a similar performance for all 3 classes. The dendrite class has the lowest precision of 0.94, indicating that this class has the highest percentage of false positives. The soma class has the lowest recall of 0.96, indicating that this class has the highest percentage of nodes which were wrongly assigned to the other 2 classes.

type / metric	precision	recall	F_1-score	support
dendrite	0.94	0.98	0.96	12883
axon	0.99	0.97	0.98	26610
soma	0.96	0.96	0.96	9395
accuracy			0.97	48888
macro avg	0.96	0.97	0.97	48888
weighted avg	0.97	0.97	0.97	48888

Table 3.1: Skeleton node-level performance of the context-based 3-class model (axon, dendrite, soma) with a radius of 40 µm and 80,000 points. The support represents the number of nodes with the respective classes included in the ground truth.

Fig. 3.7 shows visualizations of the predictions and Fig. 3.8 shows an example of the worst performing chunks which were detected during inference by inspecting the chunks with the highest loss. As shown in Fig. 3.8 some processed chunks still show many wrongly labeled nodes. This indicates a high variation in the prediction of individual samples which seems to be removed by the majority vote over multiple predictions from different chunks. Many faulty samples are located where the axon and dendritic branches

emerge from the soma. The low performance in these regions is understandable as the morphologies of axon and dendrite branches are quite similar at the initial segments of the branch and even human annotaters could have difficulties identifying the correct class of skeleton nodes in case of an unfavourably generated context.

With a skeleton node-level F_1-score performance of 0.97 the 3-class model was shown to produce reasonable results. Previous approaches using CMNs based on the multi-view approach reported F_1-scores of around 0.955 for the 3-class task [35]. However, a direct comparison between the ConvPoint and the CMN approach was not possible as the models of both approaches have been trained on different data sets. The ground truth used for the trainings of the CMNs consisted of 28 reconstructed cells, while the ConvPoint models were only trained on the 20 cells of the training set.

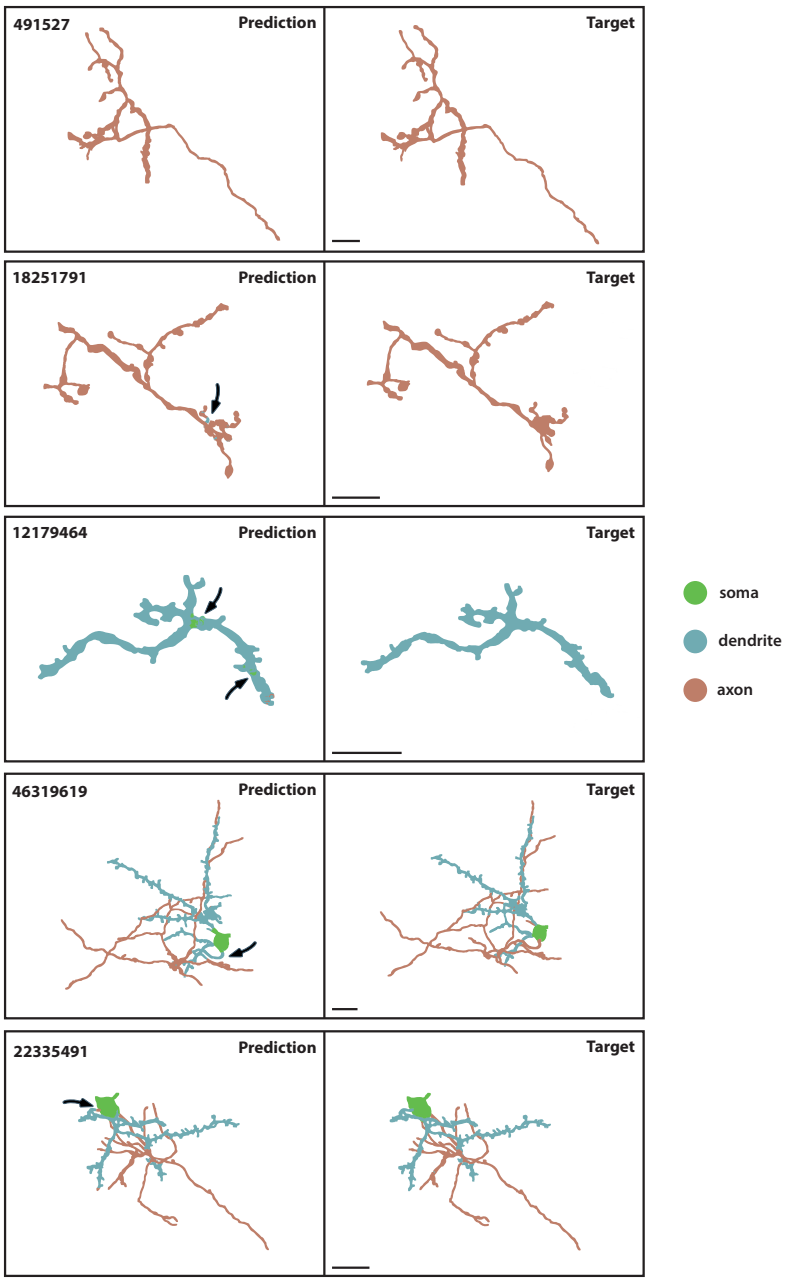

Figure 3.7: Prediction visualizations for the context-based 3-class model (axon, dendrite, soma) with a radius of 40 µm and 80,000 points. Arrows indicate faulty regions. The numbers in the upper left represent the cell id, appendix A gives more information about the respective cells. The scale bars are 20 µm.

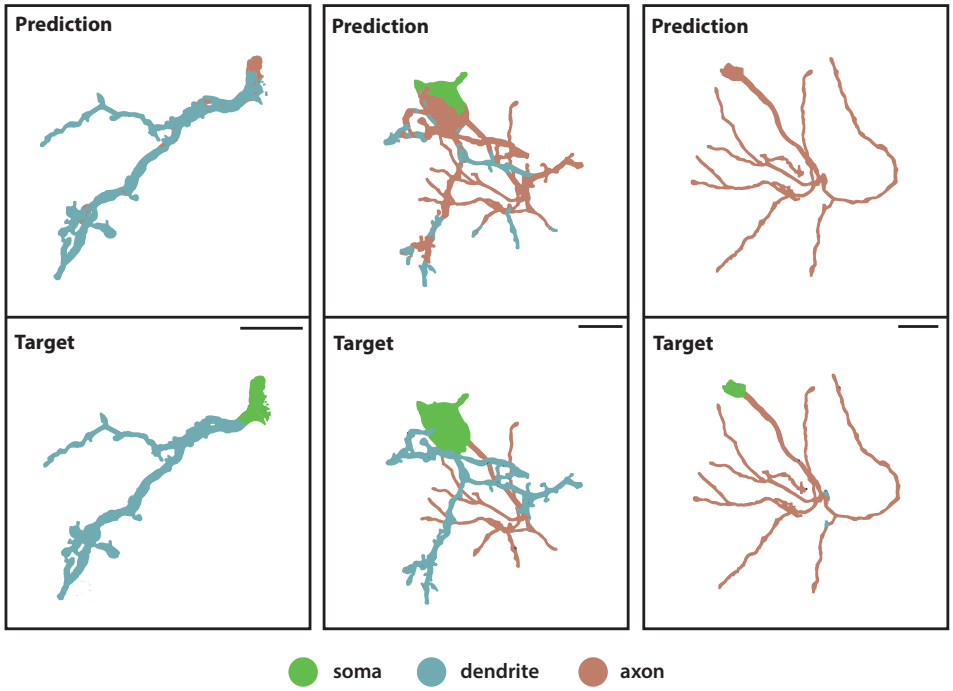

Figure 3.8: Examples for chunks with high losses processed by the context-based 3-class model evaluated in section 3.3.1. The scale bars are 10 µm.

3.3.2 Spine, bouton and terminal predictions

For the 7-class task, the context-based model with a context size of 40 µm and a point number of 100,000 was used since it showed the highest performance. As for the 3-class model in section 3.3.1, the evaluation was done by processing each cell in the evaluation set 5 times. This way, 40% of the vertices got at least one prediction. The results are shown in table 3.2. Fig. 3.9 shows visualizations of the predictions and Fig. 3.10 shows visualizations of the chunks with the highest loss during inference.

Terminals had the lowest precision of 0.59 and therefore have the highest percentage of false positives. The cell with the id 18251791 in Fig. 3.10 shows examples of these false positives. As indicated by the arrows, some boutons were wrongly labeled as terminals. Vice versa, many terminals were labeled as boutons. This is understandable because of the similar morphologies (rounded shapes) of both classes. Also, as shown in tables A.1 and A.2, terminals take up the lowest percentage of the training and evaluation set. This makes them hard to learn for the model trained on the available ground truth. This argument is also true for the spine necks and spine heads, which show even lower performance than the terminal class. With an F_1-score of only 0.13, spine necks present a major classification problem. This is also shown in Fig. 3.10 where the spine heads have relatively good predictions, but spine necks are missed at almost all locations. As shown in the examples, the point cloud representations of spine necks have a relatively high resolution, indicating that the low performance is not necessarily caused by the distortion of these filigree structures. With the lowest recall of only 0.07, spine necks have the highest percentage of nodes which get wrongly assigned to different classes and are missed at most locations. From the visualizations in Fig. 3.10, it is observed that spine necks are mostly classified as dendrites, which is explained by their similar morphologies, as both have elongated shapes, when ignoring the large difference in the diameter of these shapes. Spine heads don't have such an elongated shape, but take rather rounded forms, which might explain why heads have a much higher performance of 0.50 in contrast to the neck performance. As indicated by the arrows in Fig. 3.10, spine heads are often wrongly labeled as spine necks, explaining the low precision of 0.69 for the necks.

Interestingly, the ConvPoint model seems to be able to differentiate between axon branches and dendrites. Considering the worst performing examples in Fig. 3.10, it does not seem to be the case that many spine heads or necks are assigned to axon branches. Similarly, there are no boutons or terminals visible on dendrite branches, indicating that the model is able to use contextual information (axonal or dendritic contexts) in order to differentiate between filigree structures like boutons and spines. However, one of the

examples in Fig. 3.10 shows many wrongly assigned soma vertices on an axonal branch, indicating that some problems with context evaluation still exist.

type / metric	precision	recall	F_1-score	support
dendrite	0.75	0.98	0.85	9882
axon	0.94	0.90	0.92	20160
soma	0.97	0.95	0.96	9395
bouton	0.71	0.77	0.73	5409
terminal	0.59	0.53	0.56	1041
neck	0.69	0.07	0.13	1782
head	0.70	0.39	0.50	1219
accuracy			0.86	48888
macro avg	0.76	0.65	0.66	48888
weighted avg	0.86	0.86	0.85	48888

Table 3.2: Performance results on skeleton node-level for the context-based 7-class model with a radius of 40 µm and 100,000 points.

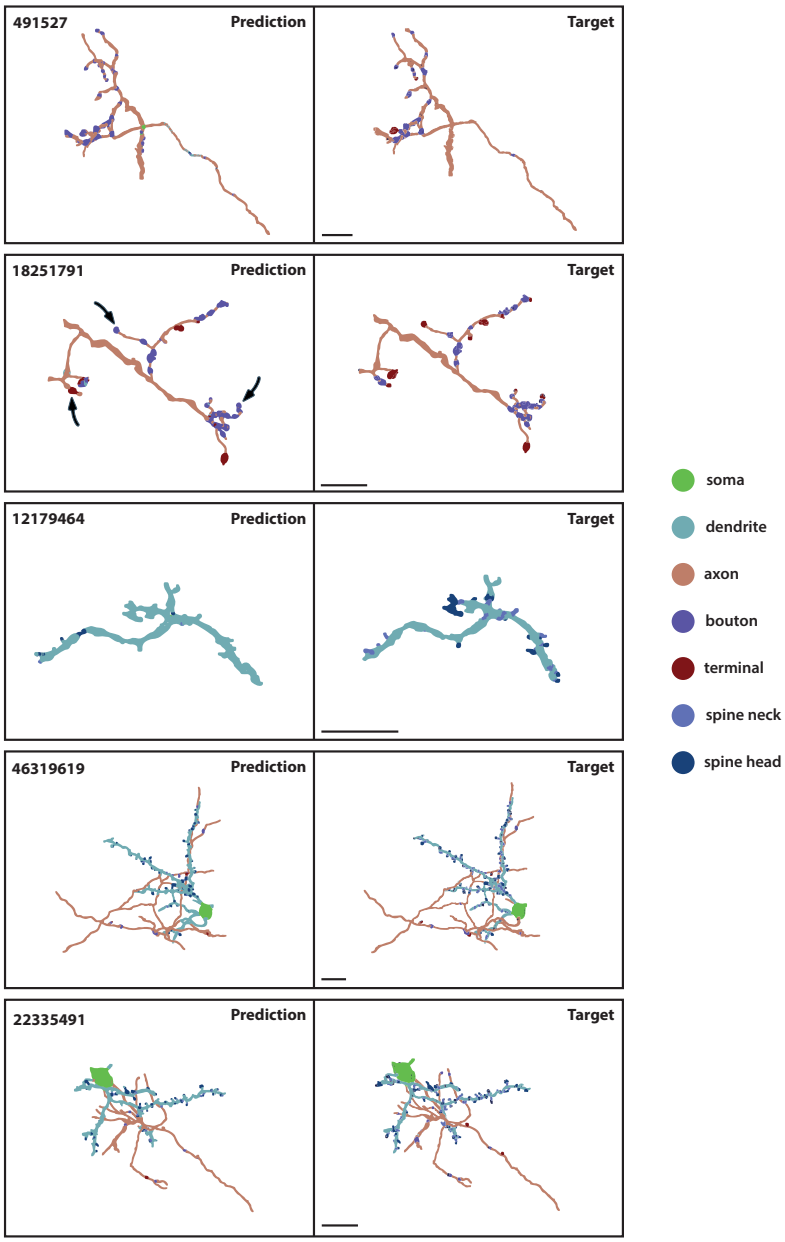

Figure 3.9: Prediction visualizations for the context-based 7-class model evaluated in section 3.3.2. The numbers in the upper left represent the cell id, appendix A gives more information about the respective cells. The scale bars are 20 µm.

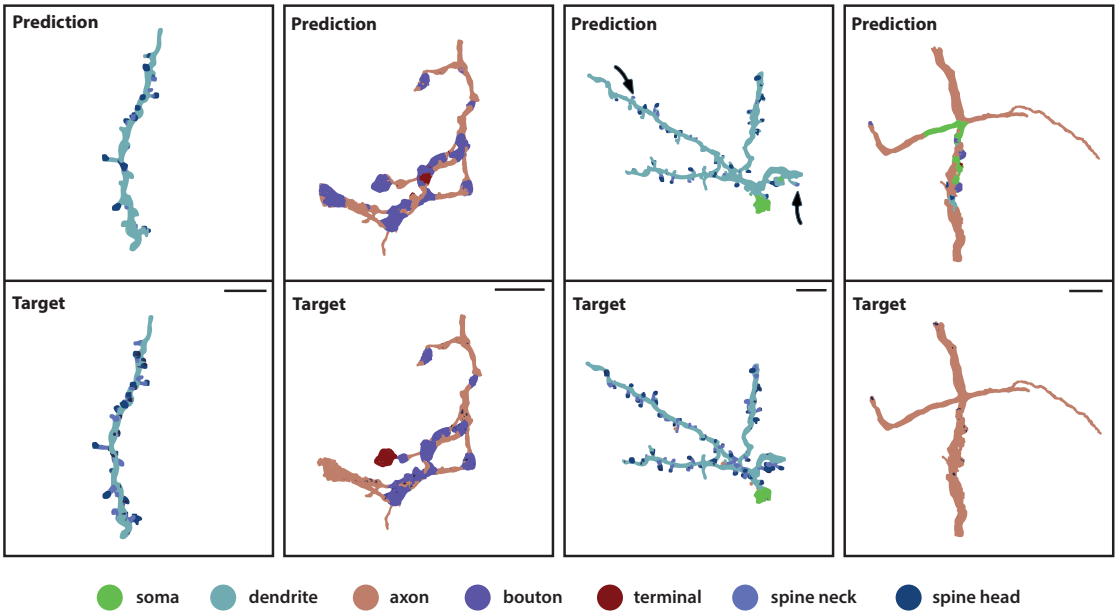

Figure 3.10: Examples for chunks with high losses, detected during the evaluation of the context-based 7-class model evaluated in section 3.3.2. Spine necks are missed at almost all locations, terminals are often mislabeled as boutons. The scale bars are 10 µm.

3.4 Effects of cell organelles and myelin

All trainings which were reported before used cell organelles. The type of the points, dependent on their location of either the cell surface of the sub-cellular structures, was indicated by a one-hot encoding in the feature domain. For trainings which included cell organelles, there were 4 feature dimensions for cell, mitochondria, vesicle clouds and synaptic junctions. For myelin there were 5 feature dimensions, using the additional dimension to differentiate between cell surface points and cell surface points with myelin. To assess the effect of these additional structures, multiple models have been trained with and without cell organelles and additional myelin encoding. Fig. 3.11 shows the F_1-score for multiple trainings with varying feature dimensionality in the inputs.

The predictive performance of the model significantly improved by including additional points of sub-cellular structures. While trainings with cell organelles reached F_1-scores of 0.64, 0.60 and 0.68, corresponding trainings without cell organelles reached scores of only 0.33, 0.34 and 0.37. This indicates that the model was able to successfully integrate and exploit the neurobiological indicators given with the described feature encoding. For example, vesicle clouds act as a strong discrimative feature, as they occur in the presynaptic structures like boutons *en passant* and terminal boutons.

Trainings which included surface points with a myelin feature reached performance of only 0.55, 0.58 and 0.64. Based on the data from this experiment, the additional myelin information seems to have a negative effect on the performance of the models. This might be caused by the encoding method, as inputs with higher feature dimensionality might be harder to learn.

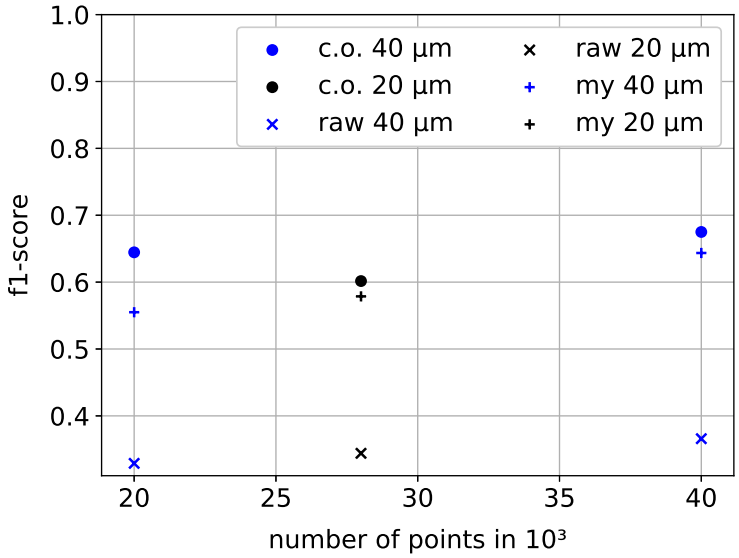

Figure 3.11: 'macro avg' performance results for assessment of effects of cell organelles and myelin on the 7-class task. 'c.o.' indicates trainings with cell organelles, my stands for trainings with cell organelles and with myelin and raw stands for trainings without both. All trainings used a context-based splitting.

Chapter 4

Conclusions and Outlook

This thesis presents a novel approach to generate a high resolution semantic segmentation of neuronal cells which were reconstructed from VEM data (Fig. 1.5). In order to efficiently handle the reconstructed cells given as a combination of a surface mesh and an underlaying skeleton, the novel framework MorphX was developed. This framework was then used to develop a processing pipeline (Fig. 2.4) in which the mesh vertices of the cells were used to represent each cell as a point cloud. The semantic segmentation of these point clouds was then generated using the recently published ConvPoint architecture [6] which is a neural network that can operate on unstructured point clouds. As the number of vertices of a cell mesh can easily exceed the maximum number of points which are processable by the graphics card, two different splitting mechanisms have been developed and integrated into the MorphX framework. One approach generated the chunks based on the actual surface area per volume and thereby guaranteed a uniform point density while keeping the total presented area fixed. (Fig. 2.2). The second approach split the cell into chunks of a fixed context size, where all vertices associated to a subgraph within a certain distance around a base node became part of the generated chunk (Fig. 2.3).

Timing results for the developed MorphX pipeline were presented in section 3.1.3. With the current version of the pipeline the processing time for most cells is below 25 seconds, excluding the time necessary for splitting. Occupying the entire cluster (see section 2.2.5) and using its 36 graphics cards and its CPUs in parallel, the processing of 1 million neurons of average size (presuming a processing time of 17 seconds per cell, including splitting) results in about 5.5 days of processing. To prepare this method for the increasing EM data sets available [18], the current implementation can be further optimized and sped up by parallelizing steps like splitting and inference. As described in section 1.5, the main problem of the multi-view approach is the large computational overhead necessary to generate, store, process and map the different views. The ConvPoint approach presented in this thesis solves this problem by operating directly on the mesh of the reconstructed cells, not requiring any other computations than splitting the

cells and processing the resulting chunks.

The point cloud chunks which were generated by the splitting mechanisms and enriched with the cell organelles were processed by a segmentation network which was taken from [6] (and which is specified in 2.1 and section 2.2.4). Using this architecture, different models have been trained on a training set of reconstructed cells (specified in section 2.4.1). The point clouds of the reconstructed cells were enriched with additional points from cell organelles like mitochondria, vesicle clouds, myelin and with synaptic junctions (Fig. 2.1). As shown in Fig. 3.11 these additional structures could successfully be exploited for the semantic segmentation of the point clouds as these cell organelles can act as strong indicators for the type of the cellular compartment they are located at. The integration of myelinated surfaces did not have a positive effect on the performance of the trained models, which must be further investigated. Future experiments could involve the testing of different type encodings, for example a multi-label approach.

In order to assess the effect of the hyperparameters used for splitting the data sets, multiple trainings with cell organelles have been performed using the two described splitting mechanisms. These trainings were then evaluated on reconstructed cells of the evaluation set (described in section 2.4.2). The reconstructed cells were processed multiple times, each turn producing predictions of only a randomly drawn subset of coherent cell vertices (dependent on the splitting mechanism). Models were trained for a 3-class (axon, dendrite and soma prediction) and for a 7-class task (expanding the targets by spine necks, spine heads, boutons and terminals). The resulting performances of these trainings (see Fig. 3.4, 3.5, 3.6) showed no clear linear dependency between the performance and the context size, sample point number or density used for splitting the cells. The data implies an overfitting to the training set, meaning that the performance on the evaluation set decreased after a certain amount of time, while the performance on the training set increased further. This is an important observation which needs to be investigated further. Due to the number of given samples in the ground truth and the time effort required for its generation, it was not possible to separate an additional validation set. A simple, but expensive solution would be to manually generate more ground truth data to increase size and variety of the training samples and to create a dedicated validation set. Until now, the only augmentations in use were random rotations to make the model invariant to this type of transformation. Future work should also focus on point-wise and patch-wise deformations of point clouds to artificially increase the ground truth size without manual effort. Increasing the number of base nodes used for the splitting yields more chunks with different contexts, thereby further increasing the variety of the ground truth.

The detailed performance analysis of the best 3-class model (Table 3.1 and Fig. 3.7) reached a F_1-score of 0.97 compared to the performance of 0.95 reported for the 3-class

task in [35] which was trained on a larger ground truth. The achieved performance for the 3-class task can further be improved by adding simple heuristics, for example a moving window averaging, which would remove any outliers on consecutive processes. Starting from the soma, the filter would for example be moved along an axon branch, smoothing out any outliers like soma or dendrite predicted nodes along the way.

The best performing 7-class model found during the parameter searches suffered from poor prediction results for fine structures with a macro average F_1-score of 0.66 (Table 3.2 and Fig. 3.9). The class-wise scores and the examples of the worst performing chunks (Fig. 3.10) showed that the prediction of spine necks was the major problem of this 7-class model, reaching an F_1-score of only 0.13 while the macro average of all classes resulted in 0.66. The increase of performance for the high resolution tasks should be the major focus of future experiments. As shown in table A.1, classes like boutons, terminals and spines take up only a small percentage of the training set. Possible measures to increase the scores therefore include the generation of ground truth which includes higher percentages of the underperforming classes. The use of specific class weights during training might also have a positive effect on the performance. Instead of training models for the prediction of all 7 classes, an performance increase could also be gained by combining different models which were trained for different tasks. This was done in [35], where models were trained on a 4-class task, including only spine heads, spine necks, dendrites and a fourth class which combined the remaining classes. The combination of such a model with another one trained for the classification of the remaining classes could yield higher performances than trying all 7 classes at once. Further adaptations in the splitting mechanisms and an even finer parameter search of the presented point density, context size and surface area could also be subject of future work.

While this thesis proposed a small improvement of existing analysis pipelines for connectomic data sets, the reconstruction of complex neural wirings or entire brains still faces major technical and methodical difficulties. However, many small steps, such as the one presented here, could result in a giant leap towards this final goal.

Appendix A

Ground truth specifications

SSV id	A_{cell}	# nodes	my	de	ax	so	bo	te	ne	he
33581058	2863.0	9678	5.3	0.0	99.6	0.0	0.4	0.0	0.0	0.0
2734465	380.0	1310	0	0.0	65.7	0.0	34.3	0.0	0.0	0.0
15933443	6349.0	20911	3.9	43.9	10.7	30.1	1.0	0.3	8.1	5.9
37558272	2357.0	7540	2.5	0.0	100.0	0.0	0.0	0.0	0.0	0.0
16113665	5176.0	17522	5.9	51.9	16.1	17.8	0.9	1.1	5.0	7.2
8003584	3427.0	11239	3.2	41.1	1.7	46.4	0.0	0.0	5.1	5.7
8339462	4848.0	18188	8.2	0.0	92.6	0.0	7.0	0.4	0.0	0.0
24414208	4692.0	15124	4.0	42.2	9.8	38.0	0.4	0.3	4.2	5.0
15982592	13949.0	43054	22.6	31.3	13.5	35.2	13.8	6.0	0.1	0.0
31967234	1421.0	6011	5.0	0.0	76.8	0.0	21.9	1.3	0.0	0.0
26501121	5724.0	18544	2.9	39.7	10.6	42.1	0.8	0.2	2.4	4.2
16096256	974.0	2266	1.0	0.0	29.4	0.0	32.6	38.0	0.0	0.0
18571264	5001.0	15203	2.5	58.5	0.9	29.8	0.0	0.0	5.0	5.8
23400450	541.0	1793	1.1	69.4	0.0	0.0	0.0	0.0	15.3	15.4
18556928	962.0	3091	0.0	0.0	28.6	0.0	70.3	1.1	0.0	0.0
34811392	947.0	2757	0	0.0	80.7	0.0	14.3	5.1	0.0	0.0
2854913	4303.0	13172	2.5	37.9	1.9	50.4	0.6	0.0	5.1	4.1
26169344	4452.0	13992	2.6	50.3	0.4	39.6	0.0	0.0	5.0	4.7
23144450	1426.0	3623	0	97.3	0.0	0.0	0.0	0.0	2.7	0.0
10919937	1009.0	2600	0.8	0.0	53.2	0.0	38.6	8.2	0.0	0.0
total	70802.0	227618	7.6	33.8	25.1	26.7	6.3	2.1	2.9	3.0

Table A.1: Specifications for cells in the training set. A_{cell} indicates the total surface area of the cells and is given in µm^2, # nodes gives the total number of nodes included in the skeleton, my stands for myelin, where the percentage is given independently from the other classes. The percentages of dendrites (de), axons (ax), soma (so), boutons (bo), terminals (te), spine necks (ne) and spine heads (he) add up to 100%.

SSV id	A_{cell}	# nodes	my	de	ax	so	bo	te	ne	he
12179464	1800.0	5000	16.9	92.1	0.0	0.0	0.0	0.0	2.8	5.0
491527	4624.0	14277	9.3	0.0	78.8	0.0	19.7	1.6	0.0	0.0
46319619	4251.0	14224	3.8	33.9	16.6	37.5	1.3	0.3	5.6	4.9
18251791	2144.0	6194	2.9	0.0	44.9	0.0	42.8	12.2	0.0	0.0
22335491	2644.0	9193	8.1	26.4	16.9	45.4	2.0	0.2	4.2	5.0
total	15463.0	48888	7.7	24.6	37.9	17.2	12.8	2.3	2.5	2.7

Table A.2: Specifications for cells in the evaluation set.

SSV id	$A_{mi}(\#mi)$	$A_{sj}(\#sj)$	$A_{vc}(\#vc)$
33581058	336.0 (159)	4.0 (15)	3.0 (3)
2734465	29.0 (26)	23.0 (35)	1.0 (6)
15933443	949.0 (339)	202.0 (558)	9.0 (27)
37558272	110.0 (86)	1.0 (4)	1.0 (3)
16113665	810.0 (349)	176.0 (405)	6.0 (19)
8003584	364.0 (147)	86.0 (232)	3.0 (8)
8339462	179.0 (178)	69.0 (151)	85.0 (123)
24414208	660.0 (263)	144.0 (347)	2.0 (8)
15982592	3015.0 (1259)	1717.0 (2116)	590.0 (270)
31967234	83.0 (85)	36.0 (79)	32.0 (85)
26501121	574.0 (248)	134.0 (279)	6.0 (16)
16096256	150.0 (77)	59.0 (53)	197.0 (44)
18571264	783.0 (276)	177.0 (405)	2.0 (6)
23400450	79.0 (25)	30.0 (68)	0 (0)
18556928	153.0 (108)	54.0 (69)	96.0 (73)
34811392	175.0 (76)	17.0 (19)	55.0 (27)
2854913	674.0 (185)	110.0 (310)	10.0 (27)
26169344	638.0 (252)	99.0 (258)	3.0 (8)
23144450	233.0 (116)	380.0 (453)	4.0 (10)
10919937	191.0 (120)	126.0 (212)	182.0 (93)
total	10187.0 (4374)	3644.0 (6068)	1286.0 (856)

Table A.3: Cell organelles of the cells in the training set. A_{mi} stands for the surface area of mitochondria, A_{sj} for synaptic junctions and A_{vc} for vesicle clouds. The number of objects is given in brackets. The areas are given in µm^2.

SSV id	$A_{mi}(\#mi)$	$A_{sj}(\#sj)$	$A_{vc}(\#vc)$
12179464	281.0 (101)	451.0 (389)	0.0 (2)
491527	589.0 (349)	93.0 (138)	148.0 (153)
46319619	542.0 (269)	123.0 (309)	6.0 (21)
18251791	436.0 (248)	147.0 (129)	471.0 (114)
22335491	416.0 (187)	73.0 (164)	4.0 (14)
total	2263.0 (1154)	888.0 (1129)	629.0 (304)

Table A.4: Cell organelles of the cells in the evaluation set. The areas are given in µm^2.

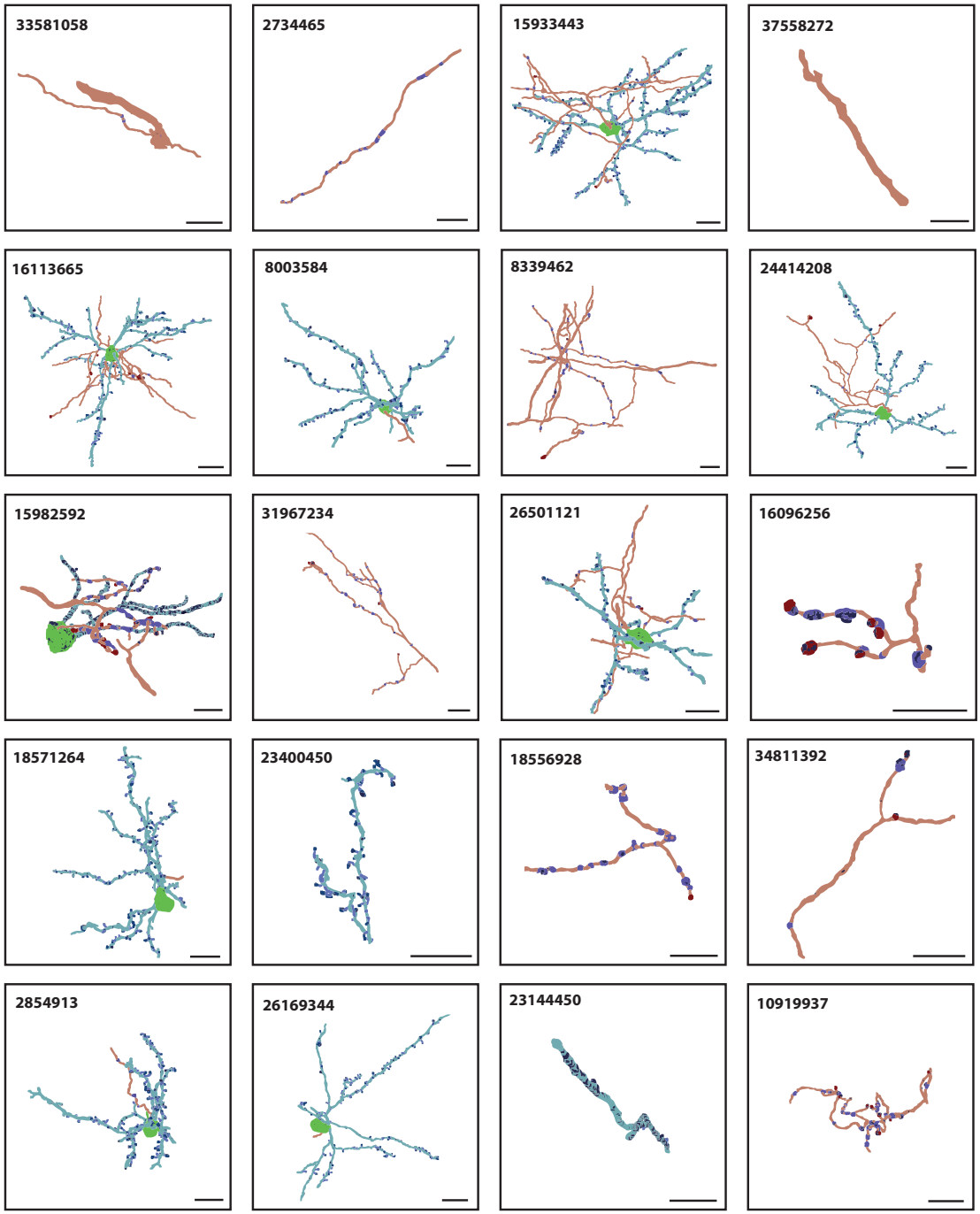

Figure A.1: Visualizations of the cells in the training set. The numbers in the upper left are the SSV ids. The color coding can be found in Fig. 2.2. The scale bars are 20 µm.

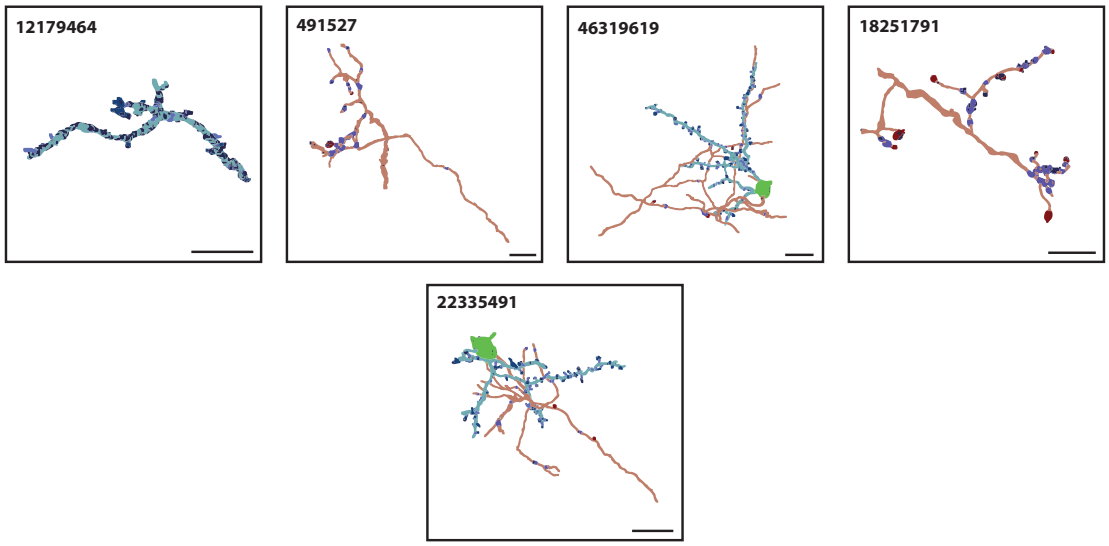

Figure A.2: Visualizations of the cells in the evaluation set. The numbers in the upper left are the SSV ids. The color coding can be found in Fig. 2.2. The scale bars are 20 µm.

Bibliography

[1] elektronn3 - a pytorch-based library for working with 3d and 2d convolutional neural networks, with focus on semantic segmentation of volumetric biomedical image data. `https://github.com/ELEKTRONN/elektronn3`.

[2] Knossos - 3d image annotation. `https://knossos.app/`.

[3] Tensorboard: Tensorflow's visualization toolkit. `https://www.tensorflow.org/tensorboard`.

[4] B. Alberts. *Molecular Biology of the Cell (Sixth Edition)*. Garland Science, 2017. ISBN 9781317563747.

[5] J. L. Blanco and P. K. Rai. nanoflann: a c++ header-only fork of flann, a library for nearest neighbor (nn) with kd-trees. `https://github.com/jlblancoc/nanoflann`, 2014.

[6] A. Boulch. Convpoint: Continuous convolutions for point cloud processing. *arXiv preprint arXiv:1904.02375*, 2019.

[7] J. Bowers, R. Wang, D. Maletz, and L. Y. Wei. Parallel Poisson Disk Sampling with Spectrum Analysis on Surfaces. *ACM Transactions on Graphics*, 29(6):1–10, 2010.

[8] W. Denk and H. Horstmann. Serial block-face scanning electron microscopy to reconstruct three-dimensional tissue nanostructure. *PLoS Biology*, 2(11), 2004.

[9] S. Dorkenwald, P. J. Schubert, M. F. Killinger, G. Urban, S. Mikula, F. Svara, and J. Kornfeld. Automated synaptic connectivity inference for volume electron microscopy. *Nature Methods*, 14(4):435–442, 2017.

[10] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[11] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun. Deep learning for 3d point clouds: A survey. 2019.

[12] C. Hammond. *Cellular and Molecular Neurobiology (Second Edition)*. Academic Press, 2001. ISBN 978-0-12-311624-6.

[13] Rana Hanocka. MeshCNN: A network with an edge. *ACM Transactions on Graphics*, 38(4), 2019.

[14] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *32nd International Conference on Machine Learning, ICML 2015*, 1:448–456, 2015.

[15] M. Januszewski, J. Kornfeld, P. H. Li, A. Pope, T. Blakely, L. Lindsey, J. Maitin-Shepard, M. Tyka, W. Denk, and V. Jain. High-precision automated reconstruction of neurons with flood-filling networks. *Nature Methods*, 15(8):605–610, 2018.

[16] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–15, 2015.

[17] G. Knott, H. Marchman, D. Wall, and B. Lich. Serial section scanning electron microscopy of adult brain tissue using focused ion beam milling. *Journal of Neuroscience*, 28(12):2959–2964, 2008.

[18] J. Kornfeld and W. Denk. Progress and remaining challenges in high-throughput volume electron microscopy. *Current Opinion in Neurobiology*, 50:261–267, 2018.

[19] J. Kornfeld, M. Januszewski, P. Schubert, V. Jain, W. Denk, and M. S. Fee. An anatomical substrate of credit assignment in reinforcement learning. *bioRxiv*, 2020. https://www.biorxiv.org/content/early/2020/02/19/2020.02.18.954354.

[20] J. Krishna Murthy, E. Smith, J. Lafleche, C. Fuji Tsang, A. Rozantsev, W. Chen, T. Xiang, R. Lebaredian, and S. Fidler. Kaolin: A pytorch library for accelerating 3d deep learning research. *arXiv:1911.05063*, 2019.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS'12: Proceedings of the 25th International Conference on Neural Information Processing Systems*, 1:1097–1105, 2012.

[22] Y. LeCun, L. Bottou, Y. Bengio, and Haffner P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11:2278–2324, 1998.

[23] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen. Pointcnn: Convolution on x-transformed points. *Advances in Neural Information Processing Systems*, 2018-December:820–830, 2018.

[24] L. Luo. *Principles of Neurobiology*. Garland Science, 2016. ISBN 978-0-8153-4492-6.

[25] J. L. Morgan and J. W. Lichtman. Why not connectomics? *Nature Methods*, 10(6): 494–500, 2013.

[26] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. *ICML*, pages 807–814, 2010.

[27] M. A. Nielsen. *Neural networks and deep learning*. Determination Press, 2018. `http://neuralnetworksanddeeplearning.com/`.

[28] E. A. Nimchinsky, B. L. Sabatini, and K. Svoboda. Structure and function of dendritic spines. *Annual Review of Physiology*, 64(1):313–353, 2002.

[29] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. De-Vito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. *NIPS-W*, 2017.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[31] C. R. Qi, H. Su, K.n Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-January:77–85, 2017.

[32] N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W. Lo, J. Johnson, and G. Gkioxari. Pytorch3d. `https://github.com/facebookresearch/pytorch3d`, 2020.

[33] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9351: 234–241, 2015.

[34] D. E. Rumelhart, Hinton G. E., and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[35] P. J. Schubert, S. Dorkenwald, M. Januszewski, V. Jain, and J. Kornfeld. Learning cellular morphology with neural networks. *Nature Communications*, 10(1):1–12, 2019.

[36] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. *Proc. IEEE International Conference on Computer Vision (ICCV)*, pages 945–953, 2015.

[37] F. Williams. Point cloud utils (pcu) - a python library for common tasks on 3d point clouds. `https://github.com/fwilliams/point-cloud-utils`.

[38] C. S. Xu, K. J. Hayworth, Z. Lu, P. Grob, A. M. Hassan, J. G. García-Cerdán, K. K. Niyogi, E. Nogales, R. J. Weinberg, and H. F. Hess. Enhanced fib-sem systems for large-volume 3d imaging. *eLife*, 6:1–36, 2017.

[39] Q.-Y. Zhou, J. Park, and V. Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

Acknowledgement

Finally, I want to thank all the people who supported me writing this thesis. In particular I would like to thank:

- Prof. Dr. Franz Pfeiffer for formally supervising this thesis and giving me the opportunity to work independently.
- Prof. Dr. Winfried Denk for giving me the chance to write my thesis in the Electrons-Photons-Neurons department of the Max-Planck Institute for Neurobiology.
- Dr. Joergen Kornfeld for proof-reading my bachelor's thesis, constructive suggestions and providing the data set used in this thesis.
- Julian Hendricks for annotating large parts of the ground truth for this thesis.
- Christian Guggenberger and the team from the Max Planck Computing and Data Facility for working in the background and maintaining the infrastructure used for this thesis.

A special thanks goes to my supervisor Philipp Schubert for giving me the opportunity to work on this interesting topic, for always answering all my questions, for providing helpful advice and guidance and for proof-reading this thesis.

A special thanks also goes to Stefan and Madlaina von Hoesslin for additional proof-reading and especially for the countless hours of fun and interesting discussions and for always bringing variety into my otherwise very routine everyday life.

I also want to thank my entire family and especially my parents for always being there for me and for raising me in a way that let me become the person I am today.

Finally I want to thank the crazy ones, the misfits, the rebels and the troublemakers without whom this world would be a far less interesting and inspiring place.

Declaration of Originality

I declare that this thesis is my own work and that, to the best of my knowledge, it contains no material previously published, or substantially overlapping with material submitted for the award of any other degree at any institution, except due acknowledgement that is made in the text.

Unterschleissheim, 31.03.2020

(Jonathan Klimesch)