Master Thesis

Error Correction in Axon Segmentation with Deep Learning

Carried out in the Visual Computing Group at Harvard University
under the supervision of Donglai Wei

Done by:
Alexandre Di Piazza

Under the direction of Mathieu Salzmann

Lausanne, EPFL, 2023

Abstract

Axon segmentation is a fundamental task in neuroimaging analysis, enabling the investigation of neuronal morphology, connectivity, and function. It is challenging due to the thin, densely packed, and often overlapping nature of axons, as well as the high anisotropy and artifact defects commonly found in neuroimaging data. As a result, existing segmentation methods often suffer from many errors. In this paper, we perform a thorough analysis of split error correction in axon segmentation, comparing different Deep Learning models. The solution is a post-processing technique, that takes as input an already computed segmentation. The first step involves utilizing high-level representations of objects with skeletons to identify potential pairs of axons that should be merged. These pairs are then subjected to a classification by a Deep Learning model to determine if any corrections are required. The resulting corrected segmentations reduce both the Variation of Information, the Adapted Rand Error, and the Expected Run Length (ERL) of the original segmentations by up to 24%, 29%, and 27%, respectively. Furthermore, for the classification, we study a point cloud approach that demonstrates comparable performance to volumetric data approaches but with significantly smaller input sizes. While some work was done on error correction in neuron segmentation, to our knowledge no work especially studied only the axons. In addition, to our knowledge, no work combined the point cloud method with the skeletonization approach. To gain deeper insights into our results, we combined different metrics, compared various models, and identified sources of errors. Such a comprehensive analysis provides valuable information and helps establish the efficacy of the method.

Keywords: Connectomics, axon segmentation, error correction

Acknowledgements

I would want to sincerely thank Donglai Wei for his excellent guidance on this project. I am very appreciative of his invaluable advice and assistance throughout, especially his in-depth understanding and proficiency in biomedical analysis and deep learning.

I also want to express my gratitude to Mathieu Salzmann for his assistance, availability, and time during the semester.

I am also grateful to my friends, who made my stay in Boston a pleasant and enjoyable experience. Their support and encouragement were instrumental in helping me to balance my academic and personal life.

Finally, I would like to express my deepest gratitude to my parents. Their unwavering support and encouragement helped me to achieve my goals and overcome challenges.

Contents

List of Figures

[1]

[1]A large proportion of the figures in this work were created using Neuroglancer, an open-source web-based tool for visualizing and annotating large-scale 3D image datasets: `https://github.com/google/neuroglancer`

List of Tables

Chapter 1

Introduction

1.1 Background

Connectomics is the study of the neural connections in the brain, typically at the level of individual neurons and their synapses. The field combines techniques from neuroscience, computer science, and physics to map and analyze the neural connections in the brain. The ultimate goal is to be able to scan a human brain and perfectly reconstruct a plan of the whole nervous system. This breakthrough would greatly enhance our understanding of neuroscience and have numerous applications in medicine, such as helping us understand the underlying causes of psychological diseases and brain disorders.

A crucial component of this research involves the 3D reconstruction of neurites from brain images. The combination of advances in image acquisition with the high-resolution images produced by Electron Microscopy (EM), and solid works on neuron segmentation, have led to very promising results surpassing human performance [3].

Currently, there is a major challenge hindering the automated reconstruction of neural circuits in the cortex, and that is the segmentation of axon instances. Axon segmentation is a sub-field of neuron segmentation that focuses specifically on detecting axons, which are neuron projections that conduct electrical impulses away from the neuron's cell body. The thin, long, and delicate branches of axons make them very prone to various types of image distortion, making them a major source of errors in neuron reconstructions.

1.2 Challenge

Considerable efforts have been devoted to refining segmentation models, yet despite achieving low error rates, a significant number of errors persist. This is largely attributed to the inherent complexity of the volumetric data and the large number of axons that need to be segmented. The datasets, achieving a very high voxel resolution ($\sim$ nm), pose multiple challenges as they are susceptible to multiple artifacts such as high anisotropy, low signal-to-noise ratio, and fragmented regions due to broken pieces. In addition, the

thin and intricate nature of axons morphology makes them especially prone to errors. These errors reduce the usefulness of the 3D reconstructions and affect the downstream analysis. In addition, manually correcting them is very time-consuming. It is thus useful to use a post-processing technique to correct those errors.

Two main types of errors occur when producing the segmentation: split errors and merge errors. A split error (left of Figure 1.1) occurs when an object is incorrectly divided into two or more separate parts. A merge error (right of Figure 1.1) is the opposite phenomena, when two or more separate parts are merged and predicted as a single object.

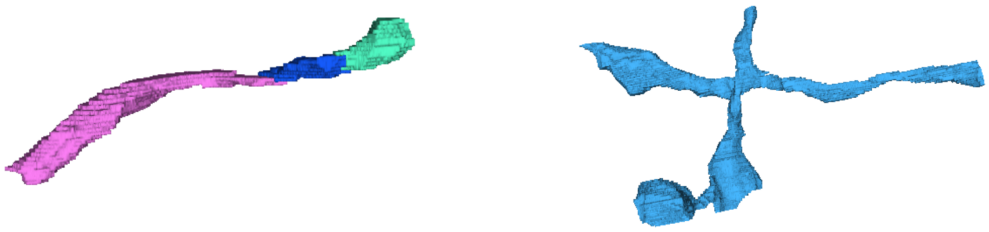


Figure 1.1: **Illustration of errors in axon segmentation**. Left: Example of a split error, where a single axon is incorrectly segmented into three separate segments. Right: example of a merge error, where two separate axons are incorrectly merged into a single segment.

In the context of connectomics, the algorithms producing the segmentation are typically tuned towards over-segmentation, producing fewer merge errors to the trade-off of producing a lot of split errors. Over-segmented outputs are preferred because a few merge errors can lead to severely incorrectly merged objects and severely impact the segmentation. In addition, merge errors are harder to detect and correct in post-processing, both for humans and automatic algorithms.

1.3 Solution and contribution

In this work, we study the correction of split errors in axon segmentations. We address error correction in two stages. First, we detect potential errors using a skeleton approach that provides a high-level representation of the 3D axon and allows us to focus on parts of the segment that are most prone to errors. In the second stage, we use deep learning models to classify whether these potential errors are actually errors or not. We evaluate the performance of four different deep learning models and two skeleton algorithms on two datasets. Specifically, we compare the full 3D image representation with a point cloud approach for the classification task. Our results show that the point cloud approach achieves among the best results while significantly reducing the computational complexity. Furthermore, we investigate the types of errors that can arise, both during the detection and classification phases.

Chapter 2

Related work

2.1 Neuron segmentation

Neuron segmentation is the process of identifying individual neurons in microscopic images of brain tissue. It plays a crucial role in neuroscience research, as it enables the study of neuronal morphology, connectivity, and function. Over the past decade, a variety of methods have been proposed for neuron segmentation.

Intermediate representation.

This approach involves using a first model to generate an intermediate representation, which is then processed by a clustering algorithm that groups pixels together to produce the final segmentation. Some methods employ a 3D U-Net or CNNs to compute an affinity-map that predicts an inter-voxel connectivity [2, 23], or boundary map reflection neuron borders [6–8]. They then use implementations of the watershed algorithm [4, 5] or graph partitioning techniques [9] to produce the final segmentation. Affinity-based models typically produce over-segmented outputs by using a low threshold with the watershed algorithm. These methods have been shown to outperform human accuracy in the task of neuron segmentation from EM images. Some additionally use agglomeration techniques such as percentile-based agglomeration [2] or hierarchical agglomeration [10, 11] to further refine the segmentation and reduce over-segmentation produced by the clustering algorithms.

End-to-end models

Unlike previous approaches, these models train a model end-to-end and produce the segmentation without the use of intermediate representation. Januszewski *et al* [13] uses CNNs recursively, segmenting one object at a time, to produce the segmentation. While achieving good results, the main trade-off is an increase in computational time. Gonda *et al* [14] uses a 3D recurrent network based on a Convolutional LSTM [15] to sequentially process each object in the image to produce the output segmentation.

2.2 Axon segmentation

Compared to neuron segmentation, less work exists on the specific task of axon segmentation. Some classical image processing algorithms have been studied, on different types of images, such as microscope images or histology images [22]. Cuisenaire *et al* [16] developed an automatic segmentation of axons with connected morphological operators. More *et al* [18] studies a semi-automated method with axon shape-based morphological discrimination, and Wang, Yi-Ying, *et all* [19] combine a watershed algorithm and fuzzy systems to segment axons. Deep Learning models have also been studied for axon segmentation. Mesbah *et all* [21], and Zaimi et all [22], propose fully automatic methods for segmenting myelin and axons from microscopy images of excised mouse spinal cord, via CNNs and Deep Convolutional Auto-Encoder. Recently in 2021, Donglai Wei *et all* [24] introduced a new volumetric dataset for axon segmentation, including EM images from the human and mouse cortex. Both images have been thoroughly proofread to provide dense 3D axon instance segmentation. They also compared state-of-the-art segmentation methods, including U-Net followed by the watershed algorithm [3].

2.3 Error correction in neuron segmentation

Errors in neuron segmentation being an important concern, a lot of research has focused on how to correct them. To our knowledge, no one focused on error corrections for axons.

First of all, errors can be corrected manually [33, 34], but it requires way too much time in order to be practical. Some research focused on how to use Machine Learning to help humans correct the segmentation by proposing errors directly [35]. Some end-to-end error correction has also been studied. Zung *et all* [39] studies the potential of using a CNN to correct errors. They use a first network for detecting potential errors, and then use another network to classify the candidates. The first network needs to be applied on each voxel of the image and is therefore computationally extremely expensive. Dmitriev12 *et al* [38] uses the same idea of a CNN to correct both split and merge errors, but reduces the search space by a large margin using a skeleton-based approach. Inputs candidates for the CNN are only joints and endpoints of the skeleton. Then, one CNN is first used to detect the merge errors by using at skeleton joints, and another CNN classifies split errors by looking at skeleton endpoints. This method is not only based on the output segmentation, but uses also the raw image as additional information for input in the CNN. In our case, we want our method to only be based on the output segmentation, and be independent of any information that lead to this segmentation, such as the original image or the affinity map created. Some also use a graph-based method [36], with nodes representing the segments and edges representing the potential segmentation errors. They use prior knowledge about neuron morphology to reduce the number of such parameters, by merging together segments that are too small to actually

be neurons, therefore reducing the number of nodes in the Graph. For the edges between nodes, they use a skeleton-based approach. Finally, they use a CNN to weigh each edge, and use a graph optimization algorithm [42], and edge contraction method, to reduce the number of nodes.

Some work focuses on correcting image defects. Berman *et all* [37] studies how to correct split errors due to the missing sections in the ER image. For one neuron on one side of the missing section, they pick candidate neurons on the other side of the missing section, based on the smallest euclidean distance. They then use a point-cloud-based CNN to pick the best candidate.

Our method combines the point cloud approach used in Berman *et all* [37] for missing section correction, and the skeleton approach used in Dmitriev12 *et al* [38] and Matejek, Brian, *et al* [36]. Whereas they study one single model, we compare the performance of different models, and analyze the types of errors that occur. It also differs from Dmitriev12 *et al* [38] as we do not use any information from the segmentation such as the original image, or the intermediate representations, to correct the segmentation. In addition, we study specifically axon segmentation.

Chapter 3

Methods

The problem of correcting split errors can be seen in two parts:

(i) Detection: find the candidate pairs of axons that are potential split errors.

(ii) Binary classification: for each pair, decide if it should be merged together.

Figure 3.1 illustrates a diagram of the correction method.

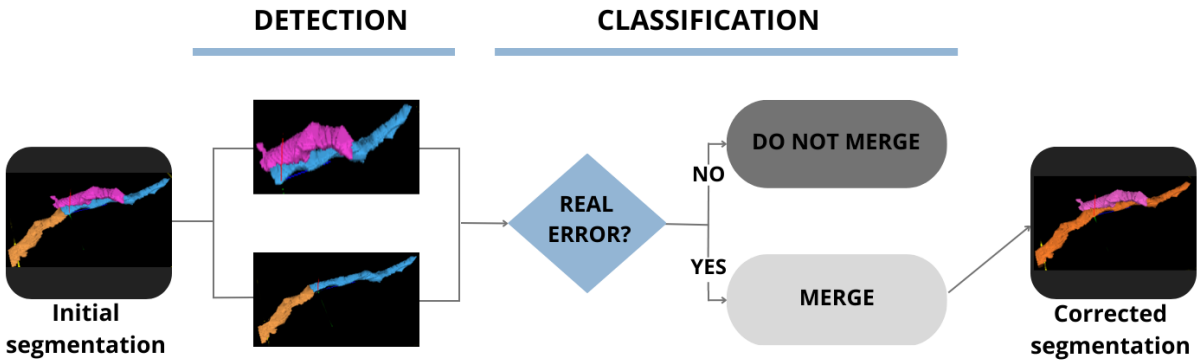


Figure 3.1: **Overview of the correction method**. Two pairs of candidate errors are detected in the first phase, and only one of them is classified as a real error and corrected to produce the final segmentation.

3.1 Datasets

We used two datasets, SNEMI3D and AxonEM-M. Both datasets used are 3D images representing axons from a different region in the mouse cortex. Each volume is fully annotated. In our case of error correction, we do not directly work with the images themselves, but with the segmentation produced from these images. The segmentation assigns each voxel an integer label that is unique for each segment. It is a case of instance segmentation with one class. We used segmentations that were computed with the

state-of-the-art method, via a U-Net producing an affinity map [3], followed by a watershed algorithm [4] and mean affinity agglomeration. Table 3.1 gives an overview of each dataset, displaying the size, resolution, and number of split errors[1] in each segmentation. The next subsections describe each dataset in more detail.

Table 3.1: **Description of the datasets.**

		Segmentation size	**Voxel resolution**	**Split errors**
SNEMI3D	Train	$100 \times 1024 \times 1024$	$30 \times 6 \times 6\ nm^3$	29
	Test	$100 \times 1024 \times 1024$	$30 \times 6 \times 6\ nm^3$	51
AxonEM-M	Train	$375 \times 2048 \times 2048$	$40 \times 14 \times 14\ nm^3$	118
	Test	$375 \times 2048 \times 2048$	$40 \times 14 \times 14\ nm^3$	89

3.1.1 SNEMI3D

The SNEMI3D (Serial Section Electron Microscopy Interconnectivity) is a publicly available dataset[2] for 3D EM images of synaptic connections in a mouse cortex. It was first proposed in 2013 for a challenge on 3D neurite reconstruction and is widely used since then. It consists of two 3D images, one for the train set and one for the test set. Each 3D image consists of 100 slices of 1024×1024 images. Fig 3.2 illustrates 2 slices of the dataset.

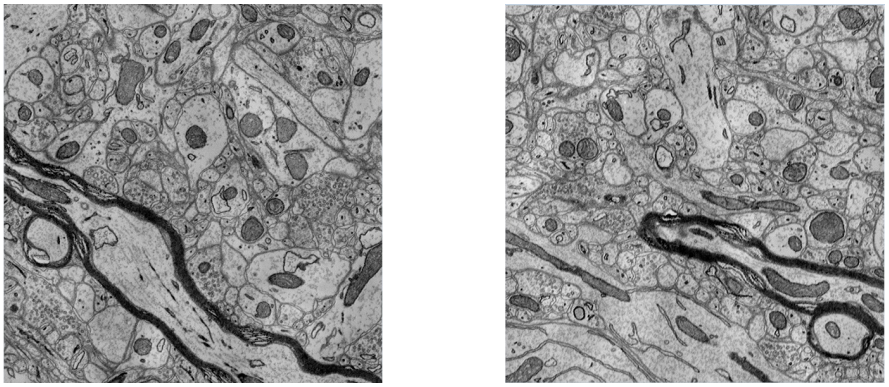


Figure 3.2: **Two slices from SNEMI3D dataset.**

Each image has a voxel resolution of $30 \times 6 \times 6\ nm^3$, respectively for the z, y and x directions[3]. The data is highly anisotropic, with a lower resolution in the z direction. All the neurites in the images have been manually labeled. The dataset was primarily developed for neurite segmentation, a neurite being any projection of a cell body. In the

[1]The number of split errors was computed by comparing the segmentation and the ground truth, excluding the non-meaningful split errors produced by broken pieces, and excluding the split errors occurring too close to the edges. Section 4.1 gives more detail.

[2]Data available on `https://zenodo.org/record/7142003#.ZAYHenbMLb0`

[3]In this paper we will always use the convention to write the spatial dimensions in this order: z, y, x.

original challenge, the goal was only to segment each neurite, without specifying if it was an axon or a dendrite specifically. However, some work has been done by neuro-scientists for further studies after the challenge. They labeled, among all the segments, the ones corresponding to axons. In this work focusing on axons, we thus only take the segments corresponding to axons, explicitly setting all the other segments to the background value. Figure 3.3 illustrates the ground truth axon segmentation of the train data, and Figure 3.4 shows examples of split errors in the segmentation.

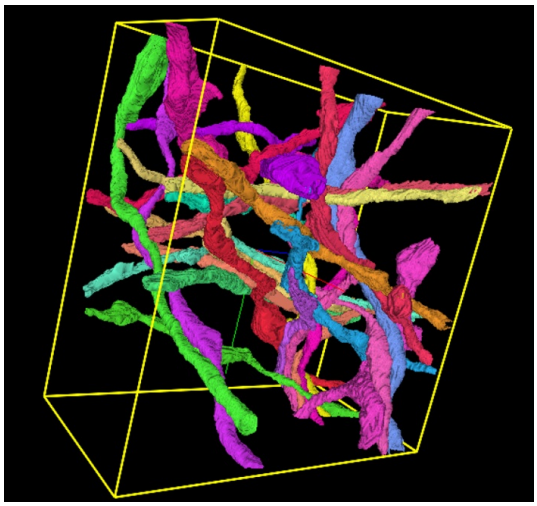
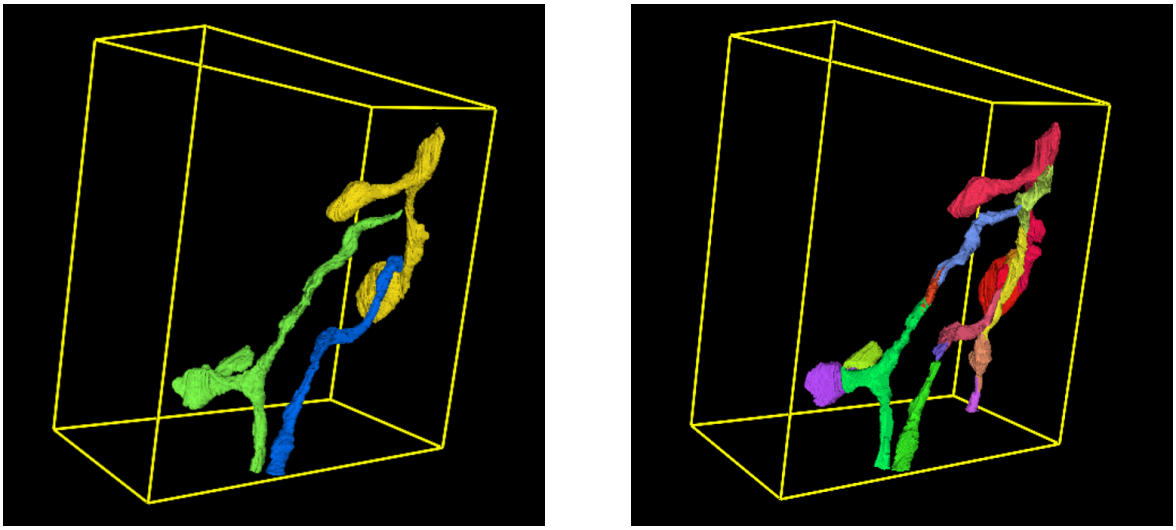


Figure 3.3: **Axon segmentation ground truth on the SNEMI3D-Train dataset**.



(a) Ground truth (b) Segmentation

Figure 3.4: **Illustration of split errors in the segmentation on the SNEMI3D-Test dataset**. Left: Ground truth, with 3 axons represented. Right: Segmentation. We can observe multiple segments predicted for each axon, indicating the presence of split errors.

3.1.2 AxonEM-M

The AxonEM dataset is a collection of electron microscopy images that was specifically created for the purpose of axon segmentation in 2021 by Donglai Wei *et all* [24]. The datasets contain two $30 \times 30 \times 30$ μm^3 EM image volumes from the human and mouse cortex. In this work, we only use the volume from the mouse cortex, which we refer to as AxonEM-M. It is a challenging dataset, where the axons often overlap with each other. It was manually annotated by experts in the field but some errors still remain. Some work was done in this project, in collaboration with Donglai Wei and neuroscientists, to correct some mistakes that remained in the ground truth. The volume is of size $750 \times 2048 \times 2048$. The first 375 slices are used as a training set, and the rest is used as a test test. Each voxel has a resolution of $40 \times 14 \times 14$ nm^3, respectively for the z, y and x directions. Like the SNEMI3D images, the data is highly anisotropic, with a lower resolution in the z direction. Even though both AxonEM-M and SNEMI3D datasets come from a mouse cortex, their distribution is different. The main difference is that axons in the AxonEM-M dataset are longer, as Figure 3.5 illustrates.

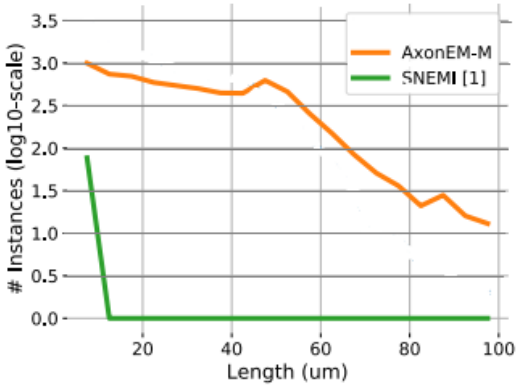


Figure 3.5: **Histogram of axon length on both datasets (adapted from Donglai *et all* [24])**. We can see that axons in the AxonEM-M dataset are longer.

Figure 3.6 illustrates the ground truth axon segmentation, and Figure 3.7 shows examples of split errors in the segmentation.

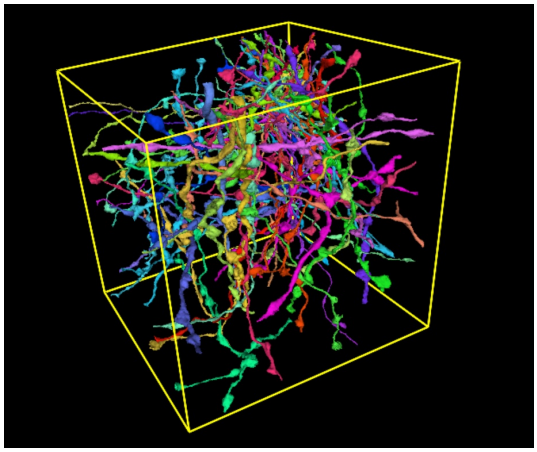
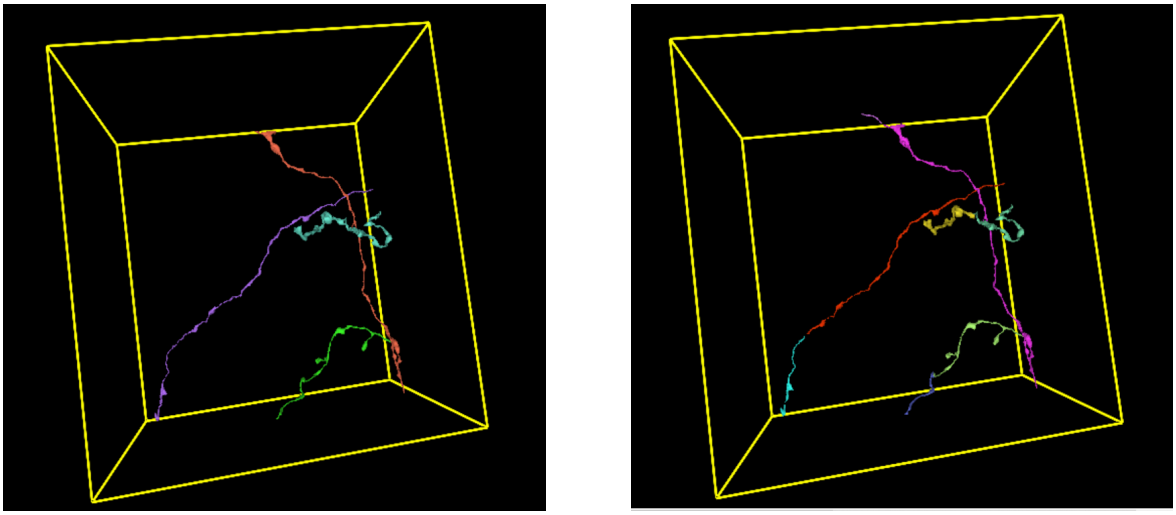


Figure 3.6: **Axon segmentation ground truth on the AxonEM-M dataset.** The figure shows a representative subset of axons from the ground truth annotations of the AxonEM-M dataset. The dataset is quite intricate, as the axons intersect and overlap with one another, presenting a complex network. The figure displays only a subset of the axons to enhance visibility and avoid overcrowding.



(a) Ground truth

(b) Segmentation

Figure 3.7: **Examples of split errors in the segmentation on the AxonEM-M dataset.** Left: Ground truth, with 4 axons represented. Right: Segmentation. We can observe multiple segments predicted for each axon, indicating the presence of split errors.

3.2 Detection: skeleton based method

The initial step in correcting errors is identifying the specific locations where they arise. It would be too computationally expensive to look at all possible pairs of axons. In the case of axons, which are characterized by their elongated and slender shape as depicted in Figure 3.8, split errors occur at the endpoints. Therefore, detecting the endpoints of

segmented axons that are in close proximity to one another is the key objective of the detection process. It allows to directly eliminate segments that touch along the main structure. On the left of Figure 3.8, we can see an example of split errors, where the fragments touch along their endpoints. However, on the right of Figure 3.8 we can see examples of axon bundles, that touch and overlap, but their intersection is not on the endpoints.

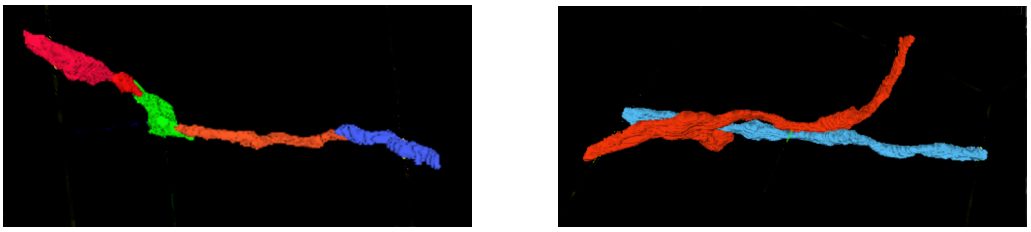


Figure 3.8: **Left:** Example of split errors with fragments touching at their endpoints. **Right:** Example of axon bundles, touching along the main structure.

A skeleton is a thin, simplified representation of an object in an image. The process of creating a skeleton, also known as skeletonization, involves reducing the object to its essential structure while preserving its topological properties. The result has the same connectivity as the original object but with a much reduced thickness. Identifying the endpoints of a 3D object can be a challenging task, but it is much simpler to locate the endpoints of a skeleton. Endpoints of the skeletons are points that have only one neighbor. Figure 3.9 shows an example of a split error and the corresponding skeleton representation. Looking at their skeleton representation, we can see that the two endpoints (shown with a circle on the image) are very close.

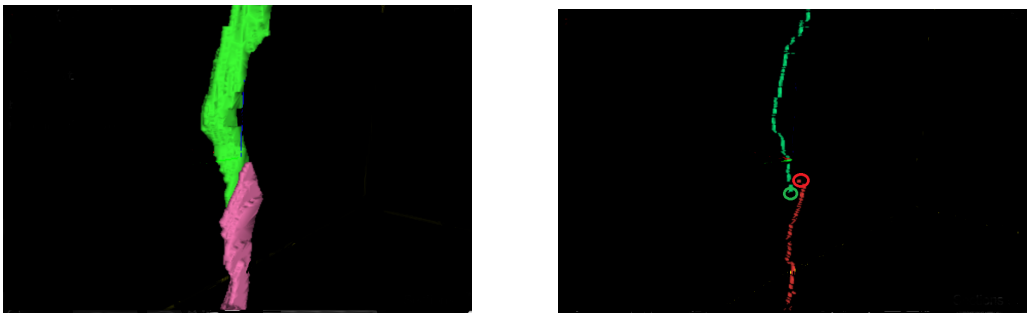


Figure 3.9: **Axons on the left, and their corresponding skeleton on the right.** The circles on the image indicate the endpoints. We can see that the two endpoints are close.

The detection method consists to compute the skeletons of neighbor segments, and detect the pairs of endpoints that are close. For the distance between two points, we use the euclidean distance, with a coefficient α to account for the anisotropy of the image

and have more weights for the pixels in z direction:

$$d(p,q)^2 = (\alpha(p_1 - q_1))^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2,\ p,\ q \in \mathbb{R}^3 \tag{3.1}$$

Pairs of endpoints that are lower to a threshold distance become candidates. We experimented with different values of threshold, as described in the results of Section 4.1.1. In Equation 3.1, we set $\alpha = \frac{\text{z-resolution}}{\text{x/y- resolution}} = \frac{30}{6} = 5$ and $\alpha = \frac{\text{z-resolution}}{\text{x/y- resolution}} = \frac{40}{14} \approx 2.9$ for the SNEMI3D and AxonEM-M datasets, respectively. Algorithm 1 summarizes the detection phase.

Algorithm 1 Detection via skeletons

Input: Segmentation
Output: List containing pairs of endpoints
Compute the skeleton of each axon
Compute the list of endpoints of each skeleton
for each each pair of neighbor skeletons $skel_i$ and $skel_j$ **do**
  for each pair of endpoints $p_m \in skel_i, q_n \in skel_j$ **do**
    if $d(p_m, q_n) <$ threshold **then**
      Add the pair to the list of candidates
    end if
  end for
end for

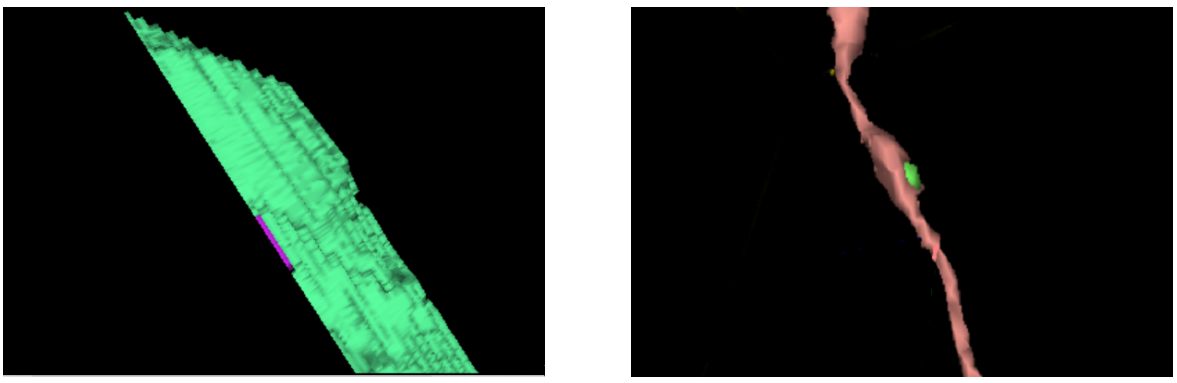
Note: In Algorithm 1, skeletons are considered neighbors when their corresponding 3D representation touch in the 3D space.

To compute the 3D skeleton, we investigated two algorithms. Scikit-image library [44] developed a method based on a thinning algorithm [45] that performs an iterative sweep over the image, removing pixels at each iteration. This process continues until the image no longer changes. The input for this algorithm is a 3D binary image with 0 representing the background and 1 representing the ID of the object we want to compute the skeleton for. The output is a 3D binary image representing the skeleton. Endpoints are found by identifying skeleton voxels that have only one neighbor. However, this algorithm is not multi-threaded. To compute all the skeletons of segments in a volume, we iteratively computed the skeleton for each segment by cropping a cuboid around it, setting the segment to 1 and the background to 0, and feeding it into the function. This reduced the input size and helped speed up the computation of all the skeletons in the image. While producing good skeletons, the algorithm can be very long on big images, due mainly to the iterative procedure. The other algorithm that we used is based on a TEASAR-derived method [46], and developed by William Silversmith *et all* [48]. The algorithm operates by first locating a root point on a three-dimensional object, and then using

Dijkstra's shortest path algorithm to sequentially trace paths through a penalty field to the furthest unvisited point. The method was designed specifically for connectomics, in order to be fast on big datasets. It has an option to be parallelized, and a parameter to limit the number of paths when computing the skeleton in order to converge faster. This algorithm was more straightforward as it allows for the direct input of the full 3D volume with segment's labels to compute all the skeletons in the image in one pass. We found the resulting skeletons to be slightly less accurate than the first method, but way faster. This method outputs a graph, with nodes representing each pixel, and edges for the connectivity. The endpoints are in this case the nodes with only one edge. For the SNEMI3D dataset, we experimented with both algorithms. For the AxonEM dataset, we used the second algorithm, faster, as the image is more volumetric.

The anisotropy of the data leads to numerous small isolated segments that can be considered noisy elements. We refer to those segments as broken pieces. Those broken pieces are characterized by a very small δz, where $\delta z = z_{max} - z_{min}$ is the difference between the maximum and minimum z coordinate of the object. We exclude objects with $\delta z < 3$ and $\delta z < 30$ in the computation of the skeletons for the SNEMI3D and AxonEM-M datasets, respectively. Similarly, we exclude those elements when computing the ground truth split errors. The values were chosen empirically based on visual expectations, with a bigger δz for the AxonEM-M where axons are longer.

Figure 3.10 illustrates examples of minor split errors due to broken pieces, that are not taken in account.



(a) SNEMI3D (b) AxonEM-M

Figure 3.10: **Examples of split errors including broken pieces.** Left: the purple part is a broken piece on the SNEMI3D dataset, with $\delta z = 1$. Right: the green part is a broken piece on the AxonEM-M dataset with $\delta z = 14$.

Correction Baseline

Our baseline for the correction process will be to use a smaller threshold distance in

Algorithm 1 and to directly correct errors by merging the two segments in the pair when endpoints are very close in space. We will refer to it as *Baseline* for the rest of the paper

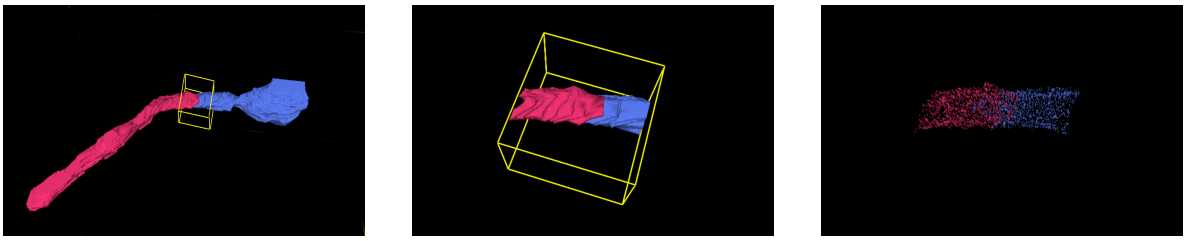
3.3 Binary classification with Deep Learning

After identifying potential errors using Algorithm 1, the next step is to verify whether each candidate pair corresponds to a real error. To achieve this, we will use a Deep Learning classifier analyzing a region surrounding each of the detected endpoints. This approach will allow us to distinguish true errors from false positives and refine the results of our error detection system. We will compare different Deep Learning models.

3.3.1 Data Preparation

Considering the volumetric nature of axon structures, it is practically infeasible to input the entire structure for each pair of candidates detected via skeletonization to the models as the corresponding input size would be too big. To circumvent this issue, we adopt a strategy where a cuboid region is chosen to be centered around the mean of the two object's endpoints. This approach provides a localized context that can be used by the classifier to distinguish between true and false errors.

We investigated two input formats. The first strategy uses the full 3D cuboid directly. The input is a 3D matrix of size $Z \times H \times H$ containing all the voxels in the cuboid. The cuboid has a different diameter in the z direction to account for the anisotropy. Each voxel of the cuboid can take 3 different values to represent the first object, the second object, and the background. The second format is a point cloud, only taking the coordinates of the points on the surface of each object. Figure 3.11 illustrates the two input formats corresponding to a pair of candidates.



(a) Axon pair of candidates detected

(b) Input format 1: 3D cuboid

(c) Input format 2: point cloud

Figure 3.11: **Illustration of the input creation.** After detecting an axon pair, we create a bounding box to capture the local context, as illustrated in (a). The resulting inputs for the classifier are represented in two different formats, shown in (b) and (c).

Each point from the point cloud is a $4D$ vector. The first 3 dimensions correspond to the z, x, and y coordinates respectively. The last dimension is a label 0 or 1, which indicates

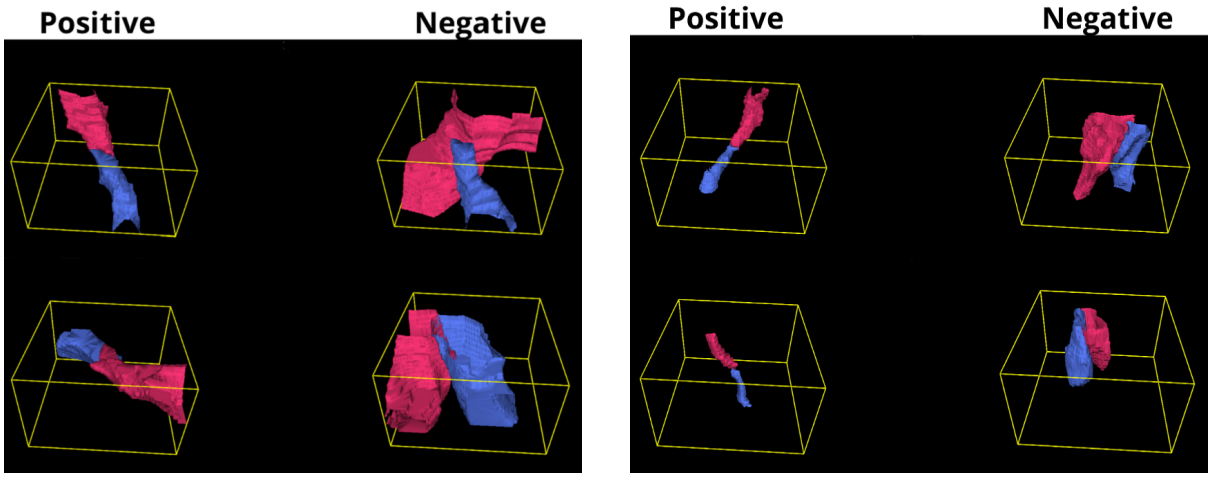
to which of the two segments the point belongs. The final input is a $4 \times 2N$ matrix, where N is the number of coordinates taken from each segment. Each spatial dimension is scaled between 0 and 1 with the Min-Max normalization:

$$x_i = \frac{x_i - min(x)}{max(x) - min(x)} \tag{3.2}$$

The N points from each object are sampled randomly from the surface, without replacement. If it happens that an object has M number of points, with $M < N$, we take all the M points and then sample with replacement until we reach N points. The corresponding point cloud thus contains duplicated points, but the cases where it happened were extremely rare.

Illustration of inputs

Figure 3.12 provides visual examples of both positive and negative samples of cuboids.



(a) SNEMI3D (b) AxonEM-M

Figure 3.12: **Illustration of cuboid inputs on the two datasets**. On positive samples, we can observe a sense of continuity in the segments, indicating that they should belong to a singular entity. On the other hand, negative samples exhibit distinct and independent objects.

Data Augmentations

We perform data augmentations including rotations, flips, and symmetries. We also add some noise by randomly setting voxels to 0 for the $3D$ cuboid, and we jitter the coordinates of the point cloud. For the $3D$ cuboid, we cannot directly use 3D rotations to preserve the anisotropy. As a result, we only perform rotations in the x-y plan. Because the normalization performed in the point cloud already loses the anisotropy, we directly perform 3D rotations on the point cloud. Moreover, for each pair of candidates, we can obtain two distinct inputs by permuting the binary label assigned to the two segments.

We also tried to use synthetic images to improve the results, but they ended up damaging the results and were therefore not used. The creation of synthetic images is described in Appendix A.

3.3.2 Model Architectures

3D CNN

A 3D CNN is the typical architecture to study 3D data and is often used to analyze medical images. The network that we use is similar to the one used in Matejek, Brian, *et al* [36]. It is made of 3 convolutional blocks, followed by two fully-connected layers and a final sigmoid function. Each convolutional block is composed of 2 3D convolutional layers with LeakyRelu activation functions and dropout layers, followed by a MaxPool operation. The pooling kernels are of size (1,2,2) to account for the anisotropy and not downsample in the z-direction.

We will refer to this model as *3D CNN* for the rest of the paper.

Siamese 3DCNN

Instead of directly feeding the 3D image containing the two objects to the *3D CNN*, the Siamese network independently feeds each object in the same convolutional blocks as in the *3D CNN* described in the previous paragraph. For an original image of size $Z \times Y \times X$ containing 0 for the background, 1 for the first object, and 2 for the second object, the image for the first object will correspond to the same original image but setting all the pixels of value 2 to 0, and similarly, the second image will set all the values of 1 to 0. The two new inputs have thus the same dimensions as the original image, but only represent a single object each. We then merge the extracted features, before feeding the results in an MLP (Multilayer Perceptron). Figure 3.13 illustrates the structure of the model. For the merging operation, we took the absolute value of the difference between the two feature vectors.

We will refer to this model as *Siamese* for the rest of the paper.

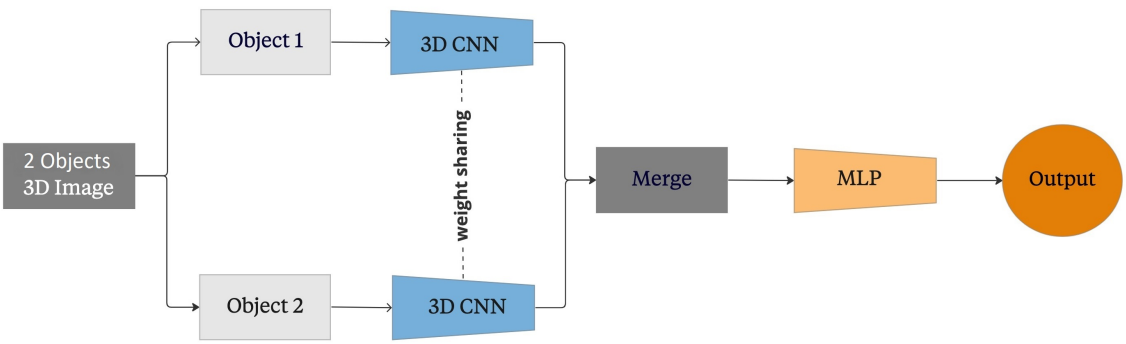


Figure 3.13: **Structure of the Siamese network.**

2.5D CNN

This model achieved first place in a Kaggle competition on detecting cervical spine fractures in 3D images, with over 800 teams. The winning solutions [41] revealed that a 3D CNN performed poorly in this task. Instead, the solution proposed using a 2D CNN for each slice of the 3D image, extracting features, and feeding them into a two-sided LSTM [17] as shown in Figure 3.14. We found this idea to be innovative and could suit our case where the lower z-resolution could adversely affect the results of a 3D CNN. We adapted their code [41] for our model, adjusting the number of input slices to our specifications.

We will refer to this model as *2.5D CNN* for the rest of the paper

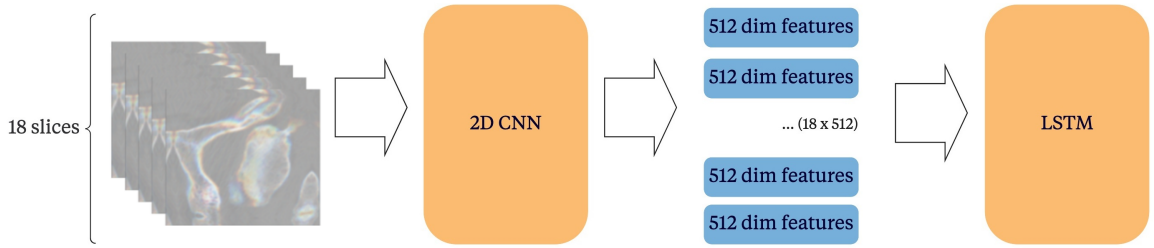


Figure 3.14: **Structure of the *2.5D CNN* (adapted from Qishen Ha [41]).**

PointNet [43]

To reason about the 3D point cloud, we use PointNet [43], a network developed by Charles R. Qi *et all* [43], commonly used for segmentation, object detection, or classification of point cloud data. The network consists of feature extraction and fully connected MLP layers that operate on individual points in the input set of points. These layers extract local features from each point using shared weights across all points. A max pooling operation is then performed over all points in the set to obtain a global feature vector representing the entire set. The global feature vector is then passed through several fully connected layers followed by an activation function to predict the output class. In the original paper developed for multi-class classification, they used a softmax activation function. In our case, we use the sigmoid activation function.

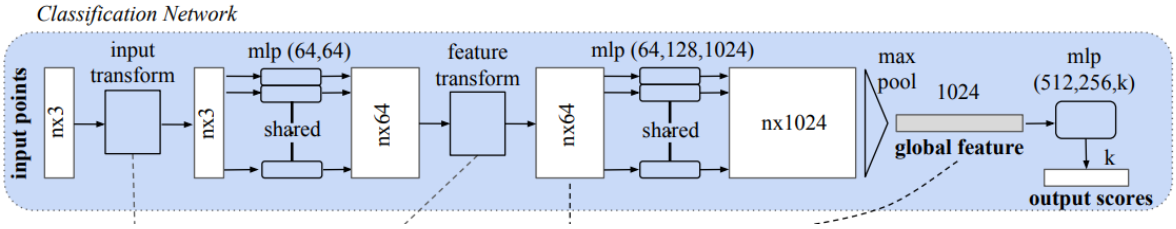


Figure 3.15: **Architecture of PointNet (adapted from Qi *et all* [43]).**

Due to the symmetry of the max-pooling operator, the network is invariant to the permutations of the points in the input, i.e :

$$f(x_1, x_2, ..., x_n) = f(x_{\pi_1}, x_{\pi_2}, ..., x_{\pi_n}),\ x_i \in \mathbb{R}^d \tag{3.3}$$

where π_i refers to any permutation. This invariance is a crucial property as the order of the points within a point cloud set does not affect the representation.

3.3.3 Training Strategy

We train the models using the weighted binary cross-entropy, with the weight for positive labels defined as $w = \frac{P}{N}$, where P and N are the numbers of positive and negative samples in the training dataset, respectively. We perform data augmentations at each epoch, and up-sample the positive labels for the SNEMI3D dataset where the data is unbalanced, by doing more augmentations for the positive labels.

3.4 Corrected segmentation

To correct the split errors in the initial segmentation, we will merge all pairs that have been identified in the detection phase and classified as positive by the classifier. This involves assigning the same label to two segments that originally had different labels. In some cases, multiple pairs that we decide to merge may share a common label. For instance, merging segments (s_1, s_2) and (s_2, s_3) creates a chain of three segments (s_1, s_2, s_3) merged into a single segment. We thus give in those cases the same label to all the segments in the chain.

3.5 Metrics

3.5.1 Evaluation of the detection phase

In the detection part, the primary objective is to identify as many actual candidates as possible, even if it results in a higher number of false candidates. This is because the ultimate goal is to correct as many errors as possible, and missing too many errors during detection can render even the best binary classification ineffective. However, if most of the actual errors are detected, even if it results in a higher number of false candidates, a highly accurate classifier can help correct most of the errors, avoiding the false splits. Therefore, the most important metric for this task is *Recall* :

$$Recall = \frac{TP}{TP + FN} \tag{3.4}$$

3.5.2 Evaluation of the classification phase

Whereas in the detection part, the goal is to have a high recall, in the case of the classification the main goal is to have a high precision:

$$Precision = \frac{TP}{TP + FP} \tag{3.5}$$

False positives create a merge error in the corrected segmentation, by merging two distinct axons into one instance. Those mistakes can severely damage the initial segmentation, instead of correcting it. As a result, we will use the F_β score, a weighted harmonic mean of precision and recall:

$$F_\beta = (1 + \beta^2)\frac{Precision \times Recall}{\beta^2 Precision + Recall} \tag{3.6}$$

A value of $\beta < 1$ weights precision higher than recall. We experimented with $\beta = 0.5$ and $\beta = 0.3$

3.5.3 Evaluation of the final corrected segmentation

To evaluate the corrected segmentation, we use three different metrics. This approach allows us to assess the results from multiple angles and determine if improvements were made across various metrics.

Adapted Rand Error

The Adapted Rand Error [51] is the metric used in the SNEMI3D competition for neuron segmentation and is based on the pairwise pixel Rand index. For a segmentation S and the ground truth G, the true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) are defined as follows:

$$TP = \sum_i \sum_{j>i} \lambda(s_i = s_j \wedge g_i = gj) \tag{3.7}$$

$$TN = \sum_i \sum_{j>i} \lambda(s_i \neq s_j \wedge g_i \neq gj) \tag{3.8}$$

$$FP = \sum_i \sum_{j>i} \lambda(s_i = s_j \wedge g_i \neq gj) \tag{3.9}$$

$$FN = \sum_i \sum_{j>i} \lambda(s_i \neq s_j \wedge g_i = gj) \tag{3.10}$$

where $\lambda(x)$ returns 1 if x is true, and 0 otherwise. For each possible voxel pair in the image, the labels of the pair (s_i, s_j) in S and the corresponding pair (g_i, g_j) in G are compared. For example, TP counts the number of voxel pairs for which $s_i = s_j$ and $g_i = g_j$. The FP, counting the number of voxel pairs for which $s_i = s_j$ and $g_i \neq g_j$, correspond to the under-segmented voxel pairs, while FN accounts for the over-segmented

voxel pairs. In our case, we exclude the background voxels in the computation. These computations, different from the usual definitions of TP, FN, FP, FN, come from the fact that we cannot directly compare the labels in the segmentation and in the ground truth as some traditional classification error metrics do. The labels in the ground truth and in the segmentation can be different even though they represent the same region. To illustrate this idea, imagine that we have an original image: I = $\begin{pmatrix} * & * & * \\ * & * & * \\ * & * & * \end{pmatrix}$, with a corresponding ground truth G = $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 2 & 2 & 2 \end{pmatrix}$, and the segmentation $S = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 4 \\ 3 & 3 & 3 \end{pmatrix}$. The segmentation S is in this case a perfect segmentation, as the clusters detected correspond to the same clusters at the ground truth G, even though they do not have the same labels.

With equations (3.7 - 3.10), we can then compute the F-score:

$$F\text{-}score = \frac{2TP}{2TP + FP + FN} \tag{3.11}$$

and finally

$$Adapted\ Rand\ error = 1 - F\text{-}score \tag{3.12}$$

The adapted Rand Error is between 0 and 1, with 0 indicating a perfect segmentation.

Variation of Information

The Variation of Information [52] is a measure of the distance between two clusterings. Let X and Y be two sets, with $X = \{X_1, X_2, ..., X_k\}$ and $Y = \{Y_1, Y_2, ..., Y_l\}$. Let also

$$n = \sum_i |X_i| = \sum_j |Y_j|$$

$$p_i = \frac{|X_i|}{n} \text{ and } q_j = \frac{|Y_j|}{n}$$

$$r_{i,j} = \frac{|X_i \cap Y_j|}{n}$$

The variation of information is then defined:

$$VI(X,Y) = H(Y|X) + H(X|Y) = -\sum_{i,j} r_{i,j} log(\frac{r_{i,j}}{p_i}) - \sum_{i,j} r_{i,j} log(\frac{r_{i,j}}{q_j}) \tag{3.13}$$

In our case, X corresponds to the segmentation, and each X_i corresponds to a voxel label in the segmentation. Similarly, Y relates to the ground truth and each Y_j corresponds to a voxel label.

The first term, $H(Y|X)$ corresponds to the conditional entropy of Y given X and is a measure of the oversegmentation. Similarly, $H(X|Y)$ is a measure of the undersegmentation. The Variation of Information is between 0 and $log(n)$, where 0 indicates a perfect segmentation

Expected Run Length (ERL)

This metric provides an estimation of the average distance along a path without errors in the reconstructed axons. It is typically measured in μm in the context of connectomics. Unlike the two previous metrics that compare the segmentation and ground truth volumes pixel-wise, this metric compares the segmentation volume with the ground truth skeleton.

A skeleton can be conceptualized as a graph, where each voxel is represented as a node, and its connectivity to other voxels in the skeleton is depicted by the edges. Each edge can thus be seen as a pair of two voxels in the volume, that are connected in the skeleton. The ERL computes the average length of correctly predicted edges. It is defined in Januszewski, Michał *et al* [13] as follows; we define :

(i) A ground truth skeleton S consisting of edges $E = \{e_1, e_2, ..., e_n\}$

(ii) An edge e is defined by a pair of 3D points $A(e)$ and $B(e)$. Each 3D point is a tuple (x, y, z)

(iii) S(e) denotes the ID of the ground truth skeleton corresponding to edge e

(iv) $Pred(e)$ denotes the ID in the predicted segmentation, corresponding to edge e in S in the ground truth. $Pred(e)$ denotes $Pred(A(e))$ or $Pred(B(e))$ when it is not ambiguous. In other words, $Pred(e)$ is the label predicted in the segmentation for edge e in the ground truth skeleton S.

Let S be a correct skeleton in the ground truth, containing a set of edges $E = \{e_1, e_2, ..., e_n\}$. We can partition the set E into "correctly reconstructed components": $CRC(S, L) = \{e : e \in E \text{ and } Pred(e) = L\}$. CRC is the subset of edges of skeleton S in the ground truth that have the same label in the segmentation. The ERL of this skeleton is then the expected size of the CRC, i.e $ERL(S) = \sum_L ||CRC(S, L)|| \frac{||CRC(S,L)||}{||S||}$, with $||S|| = \sum_{e \in S} ||e||$, where $||e||$ is the length of the edge, i.e the distance between the pair of $A(e)$ and $B(e)$, computed with the voxel resolution.
Finally, the total ERL is the expected ERL of all the skeletons, i.e $ERL(\{S_k\}) = \sum w_k.ERL(S_k)$, with $w_k = \frac{||S_k||}{\sum_i ||S_i||}$. Note that the ERL assigns a good axon segment 0 length if only one of the edges is a merge error, with two different labels at its node. As a result, the metric disproportionately penalizes merge errors. We exclude the background label 0, not computing its skeleton.

Please refer to Januszewski, Michał *et al* [13] for the complete mathematical foundations of the computation, and Donglai *et all* [24] for the implementation details, including a relaxation on the penalty on merge errors.

Chapter 4

Experimental Results

4.1 Detection results

In this part, we present the results of the detection phase.

4.1.1 Hyperparameters

The primary hyperparameter to adjust in the detection process is the threshold distance of Algorithm 1, which specifies the maximum distance between a pair of endpoints for it to be considered a potential error. We experimented with various threshold distance values on the training data, trying 40, 50, 60, and 70 on each dataset. The results of each threshold distance are presented in Table 4.1. We selected the threshold distance with the highest recall score, and if multiple thresholds attained the highest recall, we opted for the one with the lower precision.

Table 4.1: **Recall and Precision for each threshold distance.**

	Recall				**Precision**			
***thresholds*:**	**40**	**50**	**60**	**70**	**40**	**50**	**60**	**70**
SNEMI3D-Train	0.759	**1**	1	1	0.200	0.150	0.118	0.096
AxonEM-M-Train	0.814	0.864	**0.872**	0.872	0.636	0.593	0.536	0.484

The best results were achieved with a threshold distance of 50 and 60 for the SNEMI3D and AxonEM-M datasets, respectively. For the SNEMI3D dataset, augmenting the threshold above 50 only decreased the precision, as the maximum possible value of recall was reached. For the AxonEM-M dataset, the maximum recall is achieved at a threshold distance of 60. Some errors were missed by the detection because some endpoints were not detected by the skeletons. While it would be possible to increase the recall by testing additional thresholds, we were unable to explore this approach within the scope of this project due to time constraints. The AxonEM-M dataset's ground truth contained errors,

which required substantial effort to verify that our detected candidates were error-free and could serve as reliable training data for binary classification.

As explained in Section 3.2, we experimented with two different skeleton algorithms. For the AxonEM-M dataset, which is more volumetric, we used the Kimimaro package [48] due to its speed. For this algorithm, we set the maximum path to 50 trees to expedite convergence. For the 'scale' and 'const' parameters of the algorithm, we experimented with different values but ended up using the default values. The Scikit-image [44] skeletonize function required no hyperparameters. Taking advantage of the multi-thread option and using 16 cores in parallels, the Kimimaro algorithm [48] was on average 23 times faster than the Scikit-image [44] skeletonize function. For the Kimimaro algorithm [48], it took around 2 hours to compute all the skeletons of the 9000[1]segments in the AxonEM-M dataset, while it took around 1min to compute all the skeletons of the 136 segments[1] in the SNEMI3D dataset. It took around 40 min for the Scikit-image [44] skeletonize function to compute all the skeletons in the SNEMI3D dataset, and we approximated that it would have taken around 2 days to compute all the skeletons on the AxonEM-M dataset. On the SNEMI3D dataset, the skeletonize function from Scikit-image [44] achieved better results. For example, with the chosen threshold distance of 50, the Kimimaro package [48] also achieved a recall of 1, but a lower precision of 0.12.

4.1.2 Results

Table 4.2 presents the final results of the detection phase for both datasets, with the threshold distances of 50 and 60 for the SNEMI3D and AxonEM-M datasets, respectively. The results of the AxonEM-M dataset were achieved via the Kimimaro function [48], while the results presented on the SNEMI3D dataset were achieved via the skeletonize function from Scikit-image [44].

The 'pairs detected' column displays the total number of axon pairs detected by the method, which are considered potential error candidates and will be further analyzed by the classifier in subsequent steps. The 'pairs missed' column displays the number of split errors that were not detected by the method. We also provide recall and precision metrics.

[1] Not including broken pieces described in Section 3.2

Table 4.2: **Results of the detection.**

		Recall	Precision	Pairs detected	Pairs missed
SNEMI3D	Train	1	0.150	193	0
	Test	1	0.133	384	0
AxonEM-M	Train	0.872	0.536	192	15
	Test	0.876	0.429	182	11

The results for the SNEMI3D dataset indicate that the skeleton method detected all the true errors with a trade-off of lower precision. In the case of the AxonEM-M dataset, we achieved a higher precision but a lower recall. We discuss further the reasons for the low precision in Section 5.1.

4.2 Binary classification results

4.2.1 Dataset

It is important to note that the results of the detection phase will serve as the dataset for the binary classification. Table 4.3 describes the resulting datasets.

Table 4.3: **Datasets for the binary classification**. Negative samples refer to the pairs that were detected but are actually not real errors, while positive samples refer to real errors that should be corrected.

		Positive samples	Negative samples	Total
SNEMI3D	Train	29 (15%)	164 (85%)	193
	Test	51 (13%)	333 (87%)	384
AxonEM-M	Train	103 (54%)	89 (46%)	192
	Test	78 (43%)	104 (57%)	182

The training set will be relatively small, comprising less than 200 samples for training for both datasets. The dataset for SNEMI3D will be highly unbalanced, with only 15% of positive samples.

While it could be suggested to manually include the true split errors from the AxonEM-M training data into our training dataset for the binary classification, even if they were not detected by the skeletonize method, we opted not to do so. Our decision was motivated by the desire to maintain the method's scalability and automatic nature without the need for manual training data augmentation.

4.2.2 Training Setup

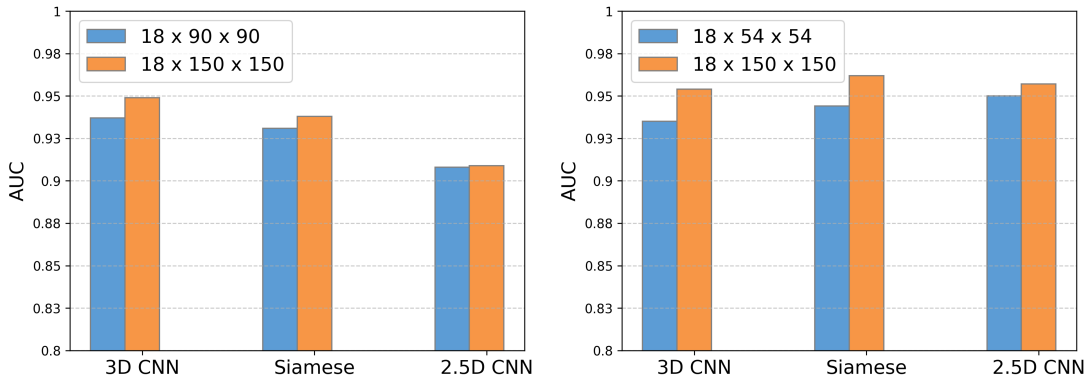
We experimented with different learning rates as well as a learning-rate finder [47], and eventually use $1e-3$ for *PointNet* [43] and $1e-4$ for the other models. To avoid memory overload, we employed a batch size of 4 and accumulated-grad batch of size 4, resulting in an effective batch size of 16. The loss was optimized via AdamW algorithm [49] with $\beta_1 = 0.9,\ \beta_2 = 0.999$ and a cosine learning rate decay [50]. We selected the final epochs based on when the validation loss began to increase, using a validation set that included 15% of the data. Dropout values of 0.2 were used for *3D CNN* and *Siamese*, and 0.3 for *2.5D CNN* and *PointNet* [43]. The models were trained on NVidia A100s GPUs.

We performed data augmentations detailed in Section 3.3.1 at each epoch. For the SNEMI3D dataset, we upsampled the positive labels by doing 15 augmentations for the positive examples, and 5 augmentations for the negative examples. For the AxonEM-M dataset, we used 15 augmentations for each image. We also used 20 validation augmentations per image to compensate for limited data, ensuring more meaningful validation results. Finally, we used data augmentation to ensemble predictions during inference by taking the average of 16 augmentations.

4.2.3 3*D* cuboid parameters

Input dimensions

Choosing the size of the bounding box around neighboring segments is a critical parameter for the input to 3D models. We evaluated two sizes of cuboid for each dataset by visually inspecting the context necessary to differentiate between pairs that belonged to the same instance and pairs that did not. For the SENMI3D dataset, we first tried a size of 18 pixels $\times$ 90 pixels $\times$ 90 pixels, preserving the anisotropy factor of 5 in the z direction. For the rest of this part, we will not specify the 'pixels' when relating to bounding box size when it is not ambiguous. We found by inspecting that sometimes, the context in the x-y plan was not enough even as a human to distinguish the two axons. We thus also explored the size $18 \times 150 \times 150$. Similarly for the AxonEM-M, we explored $18 \times 54 \times 54$ and $18 \times 150 \times 150$. The first tuple preserves the anisotropy factor of 3 of the dataset, while the second increases the context in the x-y plan. We trained and tested the models on both input sizes, and Figure 4.1 show the Area Under the Curve (AUC) achieved on the test sets. We can see that we obtained a better AUC when the models were trained with a bigger context in the x-y plan. We thus decided to finally use a cuboid of size $18 \times 150 \times 150$ for both datasets.



(a) SNEMI3D (b) AxonEM-M

Figure 4.1: **Area under the curve for the two cuboid shapes, on the test sets.** We can see that the bigger cuboid shape achieves a better AUC on all the models.

Because the resolution of the two datasets is not the same, the corresponding regions in the brain cortex are slightly different. A $18 \times 150 \times 150$ cuboid represents a region of 0.44 nm^3 for the SNEMI3D dataset, while it represents a region of 3.1 nm^3 for the AxonEM dataset. Another approach would have been to fix a real cube of diameter $d\ nm^3$ and to choose the corresponding pixel sizes for each dataset. However, as explained in Section 3.1.2 the morphologies of the axons in both datasets are different and it is difficult to know in advance which approach is superior.

In addition, we also needed to decide how to represent the input. We explored two possibilities:

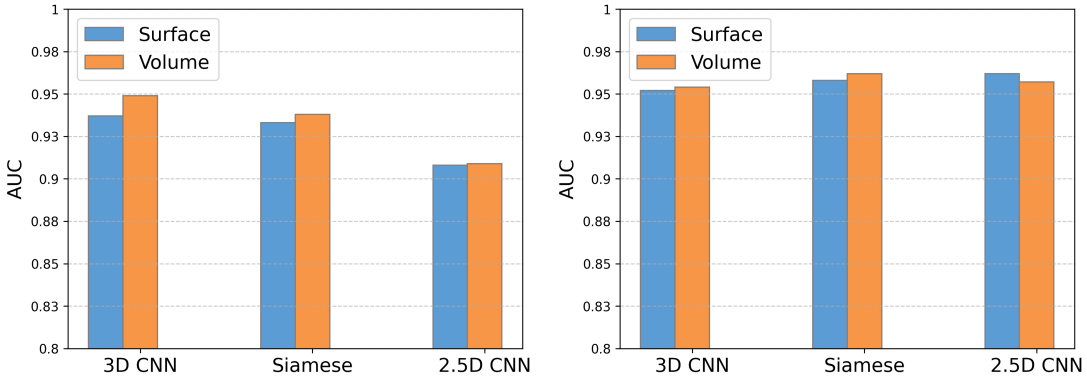
- 1 channel $(1 \times Z \times Y \times X)$, setting each segment with the values 1 or -1.
- 2 channels$(2 \times Z \times Y \times X)$, setting each segment with value 1 in each channel.

The two results achieved were very similar. Because the use of 2 channels increased both the memory and the computational complexity, we decided to use the first option with 1 channel. The size chosen for the input cuboids chosen was thus $1 \times 18 \times 150 \times 150$

Volume vs Surface

It could be argued that the surface of an object contains the most crucial information needed to analyze its shape and determine whether it should be merged with other objects. Interior points should not bring any valuable information. As a result, we investigated two ways to represent the data. In the first approach, we kept the original image containing the full volume including interior points for each object. In the second approach, we only took the voxels of the surface of each object, setting the interior points to 0. We trained and tested all the models with the two configurations, and Figure 4.2 illustrates the resulting AUC on the test sets. We can see that for the SNEMI3D dataset, inputs containing the entire volume of the objects helped the models to learn and achieve a

better AUC. On the AxonEM dataset, the results are similar for $3D$ *CNN* and *Siamese*. However, on this dataset $2.5D$ *CNN* performed better when inputs contained only the surface points of the objects.



(a) SNEMI3D (b) AxonEM-M

Figure 4.2: **Area under the curve comparison for the two input configurations.** The 'Surface' label corresponds to an image containing only the surface voxels excluding the interior points. The 'Volume' label corresponds to the full object.

Based on the previous analysis, Table 4.4 summarizes the input configurations that achieved the best results.

Table 4.4: **Best input configurations for the 3D models.**

	Model	**Input configuration**
SNEMI3D	2.5D CNN	$1 \times 18 \times 150 \times 150$ - Volume
	3D CNN	$1 \times 18 \times 150 \times 150$ - Volume
	Siamese	$1 \times 18 \times 150 \times 150$ - Volume
AxonEM-M	2.5D CNN	$1 \times 18 \times 150 \times 150$ - Surface
	3D CNN	$1 \times 18 \times 150 \times 150$ - Volume
	Siamese	$1 \times 18 \times 150 \times 150$ - Volume

4.2.4 Parameters PointCloud

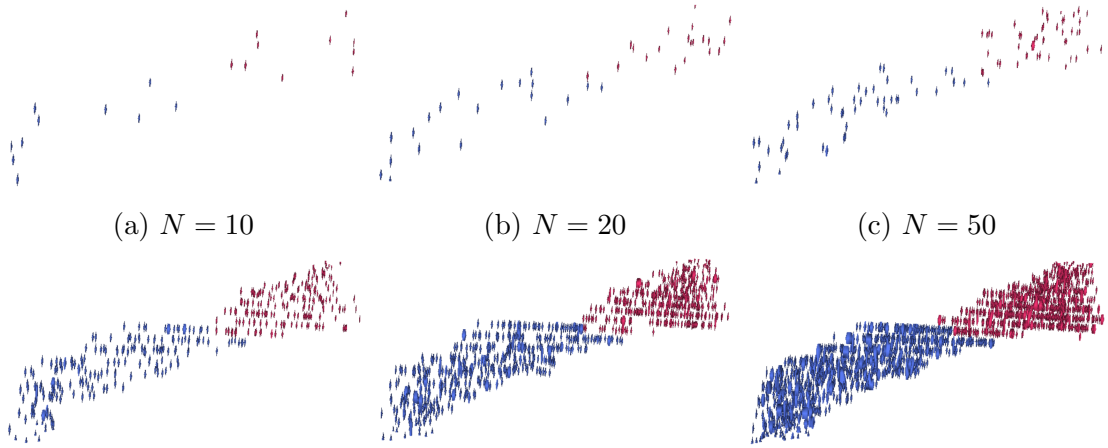
There are two main parameters that need to be chosen with the point cloud approach:

(i) The number of points N to sample from each object.

(ii) The size of the region where to sample the points.

We tried different values for N, each time training and testing with the defined value. Figure 4.3 illustrates how an input pair looks like for different value points, N referring to the number of points per object. The test set for the SNEMI3D and AxonEM-M datasets

had an average of 9800 and 2600 points, respectively, distributed across the full surface of the objects.

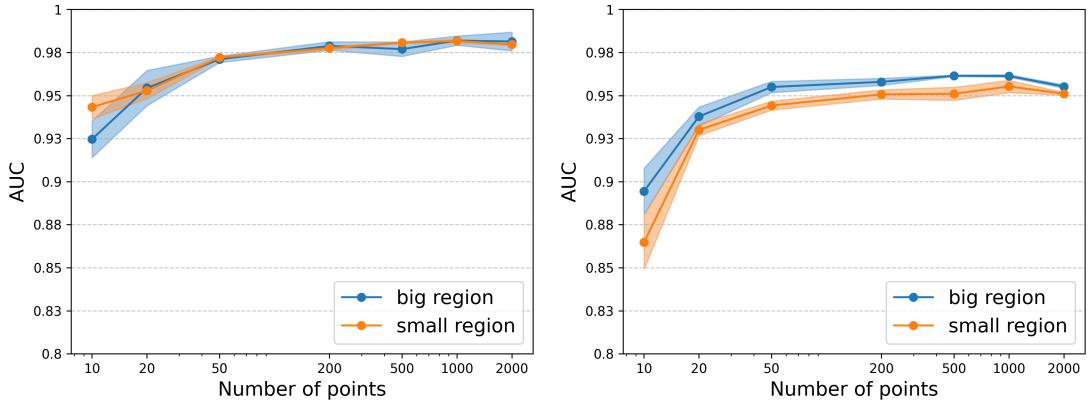


(a) $N = 10$ (b) $N = 20$ (c) $N = 50$

(d) $N = 200$ (e) $N = 500$ (f) $N = 1000$

Figure 4.3: **Point cloud for an input pair for different values of N.**

Unlike 3D models, where increasing the region size directly results in larger inputs, point clouds offer the flexibility to sample from larger regions without increasing the input size. To explore this, we experimented with two different sampling regions to extract points from: the first region is a cuboid of the same dimensions as the one used for 3D models, with a size of $18 \times 150 \times 150$; the second region is twice as large in all the directions, sampling from a cuboid with dimensions of $36 \times 300 \times 300$. Figure 4.4 illustrates the AUC vs the number of points N, for both datasets and both region's context. During inference, our model randomly samples points from each object, introducing some degree of randomness in the results. To account for this variability, we have included error bars in the figure to illustrate the standard deviations. We can see that the models achieved a high AUC even with a small number of points.

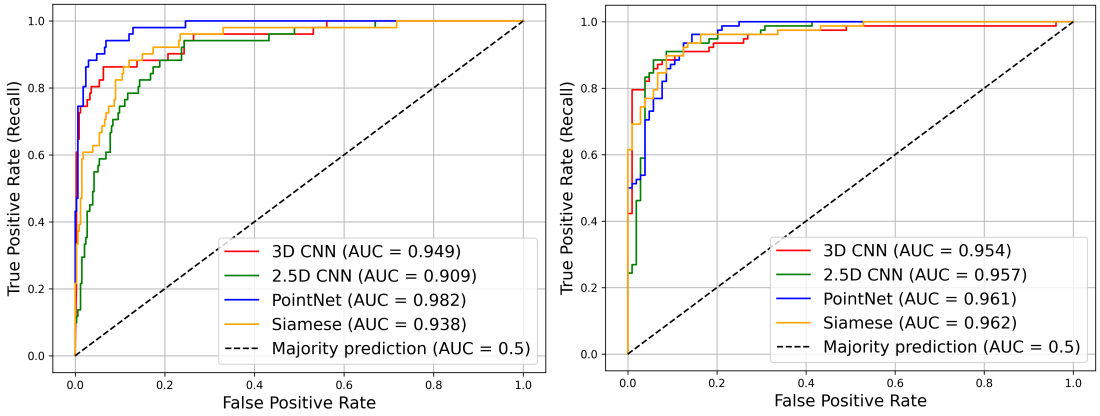


(a) SNEMI3D (b) AxonEM-M

Figure 4.4: **AUC vs Coordinates for each dataset with PointNet [43]**. The small region refers to the $18 \times 150 \times 150$ context, while the big region refers to the $36 \times 300 \times 300$ context. The error bars represent the standard deviations. We can see that on the SNEMI3D dataset, increasing the context did not improve the results. However, a bigger context was helpful on AxonEM-M.

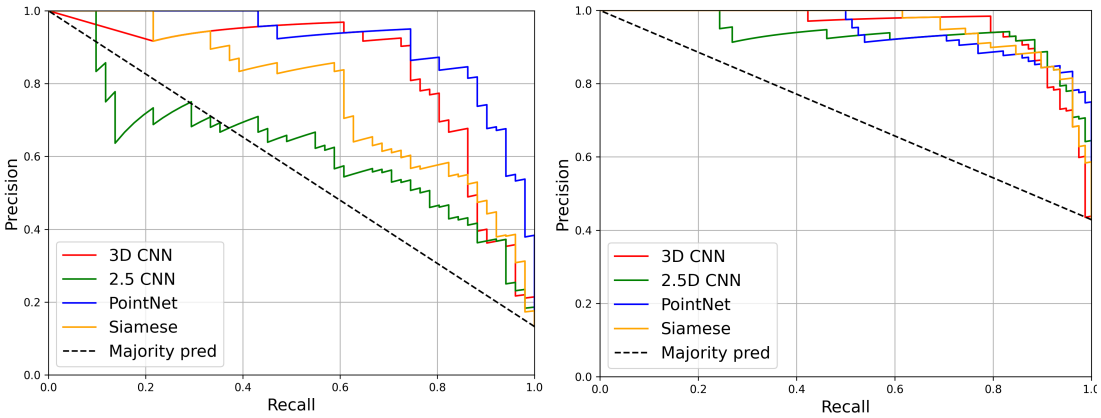
4.2.5 Results

We present the best performance of each model on both datasets. For the cuboid inputs, we give the best results corresponding to the configurations of Table 4.4. For *PointNet* [43], following Figure 4.4, we took $N = 1000$ and the small context for the SNEMI3D dataset, and $N = 500$ with the bigger context for the AxonEM-M dataset. Figure 4.5 shows the receiver operating characteristic curves (ROC) for all the models on the test set of both datasets, while Figure 4.6 displays the Precision-Recall curves.



(a) SNEMI3D (b) AxonEM-M

Figure 4.5: **ROC curve for all models**. The legend also displays the AUC for each model. The results are computed on the test set of each dataset. The majority prediction lines refer to a model predicting only the majority label.



(a) SNEMI3D (b) AxonEM-M

Figure 4.6: **Precision-Recall curves for all models**. The results are computed on the test set of each dataset. The majority prediction lines refer to a model predicting only the majority label.

The *3D CNN* and *PointNet* [43] both perform very well on both datasets, with an AUC above 0.945 in both cases. On the SNEMI3D dataset, *2.5D CNN* and *Siamese* achieve poorer performances. Being a very unbalanced dataset with 88% of zeros, we can more clearly see it on the Precision-Recall curve of Fig 4.6, where the *2.5D CNN*'s performance is close to the majority prediction referring to a prediction of only zeros. However, *2.5D CNN* achieves among the best results on the AxonEM-M dataset. It is unclear why there was such a difference in performance on the two datasets for this model. One possibility is that the model struggled to learn from the highly unbalanced data in SNEMI3D. Alternatively, it could be that the distribution of the AxonEM-M dataset, with its longer axon shapes and different voxel resolution, was better suited for the model's architecture. Overall, the Siamese structure of the *3D CNN* does not seem to bring any value.

Each model gives a continuous output between 0 and 1. We need to threshold this value to give the final prediction. We set the threshold based on the best F_β(Equation 3.6) on the validation data. As the validation data was small, we used 30 augmentations for each sample for this evaluation. We experimented with $\beta = 0.3$ and $\beta = 0.5$ on the F_{Beta} score and found that a value of $\beta = 0.3$, weighting more precision, resulted in better corrected segmentations. We thus set $\beta = 0.3$ to find the thresholds of each model, which are presented in Table 4.5

Table 4.5: **Threshold values used on each model and dataset.**

	2.5D CNN		**3D CNN**		**Siamese**		**PointNet [43]**	
	SNEMI	AxonEM	SNEMI	AxonEM	SNEMI	AxonEM	SNEMI	AxonEM
Thresholds	0.79	0.80	0.82	0.78	0.79	0.83	0.81	0.72

For the minimal euclidean distance in the *Baseline*, we set 10 for the SNEMI3D and 20 for AxonEM-M. These values were based on the best $F_{0.3}$ score on the training set. Tab 4.6 presents the final predictions.

Table 4.6: **Final predictions of all the models.** The SNEMI3D and AxonEM-M contained 88 % and 57 % of zeros, respectively. $F_{0.3}$ refers to the F_β score with $\beta = 0.3$.

	Model	**Accuracy**	**Recall**	**Precision**	$\boldsymbol{F_{0.3}}$
SNEMI3D	Baseline	0.836	0.431	0.393	0.396
	2.5D CNN	0.909	0.451	0.767	0.725
	3*D* CNN	0.958	**0.765**	0.907	0.893
	Siamese	0.914	0.608	0.704	0.695
	PointNet [43]	**0.959**	0.745	**0.936**	**0.917**
AxonEM-M	Baseline	0.742	0.718	0.691	0.693
	2.5D CNN	**0.907**	0.833	**0.942**	**0.932**
	3D CNN	0.901	0.859	0.905	0.901
	Siamese	0.890	0.897	0.854	0.857
	PointNet [43]	0.901	**0.910**	0.866	0.869

As anticipated by the results shown in Figures 4.6 and 4.5, *PointNet* and *3D CNN* are among the top-performing models on both datasets. *2.5D CNN* only achieves strong results on the AxonEM-M dataset. Notably, setting high thresholds to optimize the $F_{0.3}$ resulted in higher precision than recall, as expected. Finally, while the *Baseline* algorithm achieves reasonable results on the AxonEM-M dataset, its performance is poor and below the majority prediction threshold (accuracy of 0.836 while 88% of the samples are zeros) on the SNEMI3D dataset. This outcome was expected, given that the *Baseline* directly follows from the results shown in Table 4.2, with a lower euclidean distance for the decision. The fact that the detection created too many false positives on the SNEMI3D dataset led to poor final results for the *Baseline*.

4.2.6 Models complexity

Table 4.7 summarizes the model complexity in terms of trainable parameters and number of multiply-accumulate operations (MACs) per sample. Notably, *PointNet* [43] exhibits superior computational efficiency compared to the other models. Specifically, it requires 2.5x less MACs per sample than *3D CNN*, 5x less than *Siamese*, and 41x less than *2.5D CNN*. Moreover, it has a substantially smaller number of parameters, requiring 7x less than *3D CNN* and 15x less than *2.5D CNN*. Additionally, the input size for *PointNet* is significantly smaller than the other models. *Siamese* and *3D CNN* have the same number parameters as they use the same architecture, but the *Siamese* has twice the number of

MACs per sample because the two objects are independently fed into the model.

Table 4.7: **Models complexity**. MACs stands for multiply-accumulate operations. 'M' stands for million and 'G' stands for billion.

Models	Input dimensions	Trainable parameters	MACs / sample
2.5D CNN	$18 \times 150 \times 150$	25.0 M	23.7 G
3D CNN	$18 \times 150 \times 150$	11.1 M	1.42 G
Siamese	$18 \times 150 \times 150$	11.1 M	2.84 G
PointNet [43]	4×4000	1.6 M	0.57 G

4.3 Final impact on the corrected segmentation

To determine if the correction has improved the initial segmentation, we compare both the initial segmentation and the corrected segmentation to the ground truth using the three metrics outlined in Section 3.5. Additionally, we present the results for the best possible correction, where all the errors are detected (recall of 1 in the detection phase) and the classifier achieves 100% accuracy. This ideal scenario, which we refer to as *Perfection* in Tables 4.8 and 4.9, illustrates the theoretical maximum performance achievable by our method.

Pixel-wise metrics

We first present the results on the pixel-wise metrics: the Adapted Rand Error (ARAND) and the Variation of Information (VI). We compute the raw values and the relative improvement of the corrected segmentation over the initial segmentation for each metric:

$$decrease_{ARAND} = \frac{ARAND_{initial} - ARAND_{corrected}}{ARAND_{initial}} \times 100 \tag{4.1}$$

$$decrease_{VI} = \frac{VI_{initial} - VI_{corrected}}{VI_{initial}} \times 100 \tag{4.2}$$

$VI_{initial}$ corresponds to the VI of the initial segmentation, while $VI_{corrected}$ corresponds to the VI of the corrected segmentation. Table 4.8 shows the results for all the models, the initial segmentation, and *Perfection*. For the Variation of Information, as described in Equation 3.13, it can be decomposed into two parts: VI Merge corresponding to $H(Y|X)$ and VI Split corresponding to $H(X|Y)$. They measure the undersegmentation and oversegmentation, respectively.

Table 4.8: **Results on the corrected segmentation**. The best models reduce the Adaptive Rand Eror (ARAND) and the Variation of Information (VI) by 24% and 28.6 % on the SNEMI3D dataset, and by 9.9% and 27.7 % on the AxonEM-M datasets. Negative values in the decrease % indicate that there is an increase, reflecting a degradation of the initial segmentation.

	Model	ARAND↓(dec)	TOT VI ↓(dec)	VI Split ↓	VI Merge ↓
SNEMI3D	**Initial Seg**	0.192 (0%)	0.97 (0%)	0.912	0.058
	Perfection	0.075 (60.9%)	0.502 (48.2%)	0.442	0.060
	Baseline	0.635 (-230%)	2.38(-145%)	0.670	1.71
	2.5D CNN	0.200 (-4.2%)	0.967 (0.3%)	0.762	0.205
	3D CNN	0.155 (19.3%)	0.739 (23.8%)	0.607	0.132
	Siamese	0.183 (4.70%)	0.823 (15.2%)	0.635	0.188
	PointNet [43]	**0.146 (24.0%)**	**0.693 (28.6%)**	**0.590**	**0.103**
AxonEM-M	**Initial Seg**	0.0151 (0%)	0.0332 (0%)	0.0281	0.0051
	Perfection	0.0122 (19.2%)	0.0188 (43.4%)	0.0136	0.0052
	Baseline	0.0181 (-19.9 %)	0.0345 (-3,9%)	0.0194	0.0151
	2.5D CNN	**0.0136 (9.9%)**	**0.0234 (29.5%)**	0.0167	**0.0067**
	3D CNN	0.0140 (7.3%)	0.0247 (25.6%)	0.0169	0.0078
	Siamese	0.0147 (2.6%)	0.0266 (19.9%)	0.0160	0.0106
	PointNet [43]	0.0142 (6.0%)	0.0255 (23.2%)	**0.0157**	0.0098

The correction successfully improves the segmentation in terms of the two metrics for all the deep learning models except *2.5D CNN* on the SNEMI3D dataset which had a poor performance. Notably, *PointNet* [43] achieves the best performance on the SNEMI3D dataset, reducing the ARAND and VI by 24% and 28.6 %, respectively. It is *2.5D CNN* that produces the best results on the AxonEM-M dataset. As expected, the relative performance of each model is directly linked to the results on the binary classification; *PointNet* [43] and *CNN* achieve among the best corrections on both datasets. Decomposing the variation of information into its two components provides insight into where the reduction occurs. The VI Split decreases, indicating a reduction in oversegmentation, but this results in an increase in VI Merge. As the reduction in VI Split is greater than the corresponding increase in VI Merge, the total variation of information is reduced. This phenomenon is intrinsic to the method, which only focuses on correcting split errors and can only reduce VI Split and increase the VI Merge. The key is that the reduction in VI Split should outweigh the increase in VI Merge. Even *Perfection* induces a slight increase in VI Merge. Moreover, the Baseline damages the original segmentation instead of correcting it due to the creation of too many merges. This highlights the challenge of improving an already good segmentation, where a too simple model fails. We can

also notice an order of magnitude in the metrics between SNEMI3D axonEM datasets. The initial segmentations are more accurate on the AxonEM-M dataset.

ERL

The ERL measures the average error-free length of the reconstructed axons. When computed on the ground truth, it reflects the average length of the axons. In our case, it is expressed in μm. Table 4.9 presents the ERL on the ground truth, the initial segmentation, *Perfection*, and all the models. The percentage value shown in Table 4.9 has a different meaning than in the previous table. In this case, it represents the fraction of the ERL of the ground truth. Specifically, these values provide insight into the average percentage of the ground truth axon that is accurately reconstructed, providing a useful measure of the effectiveness of each model in capturing the true axon morphology.

Table 4.9: **ERL results on the corrected segmentation**.

	Model	**ERL ↑(μm) (%)**
SNEMI3D	**Ground truth**	10.995 (100%)
	Initial Seg	4.893 (44.5%)
	Perfection	7.520 (68.4 %)
	Baseline	2.263 (20.6%)
	2.5D CNN	4.434 (40.3%)
	3D CNN	6.086 (55.3%)
	Siamese	5.925 (53.9%)
	PointNet [43]	**6.217 (56.5%)**
AxonEM-M	**Ground truth**	30.526 (100 %)
	Initial Seg	29.852 (97.79 %)
	Perfection	30.118 (98.66 %)
	Baseline	29.713 (97.34%)
	2.5D CNN	**30.014 (98.32%)**
	3D CNN	29.991 (98.25%)
	Siamese	29.986 (98.23%)
	PointNet [43]	29.913 (97.99%)

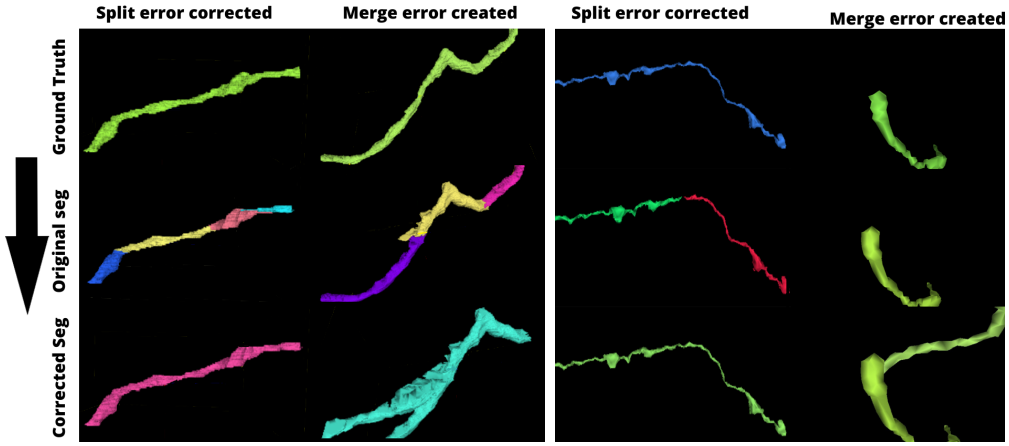
Similarly as Table 4.8, the correction successfully improves the segmentation in terms of the ERL for all the models, except *2.5D CNN* on the SNEMI3D dataset. *PointNet* [43] and *2.5D CNN* again achieve the best results on SNEMI3D and AxonEM-M respectively. On the SNEMI3D dataset, we increase the ERL% by more than 10 points with *PointNet* [43]. On the AxonEM-M dataset, the ERL% is already extremely very close to the best possible value of 100%. The deep learning models still increase the performance even though the absolute difference is smaller. The lower raw values for the ERL on

SNEMI3D come directly from the fact that the axons are smaller in this dataset. Just as pointed out in the analysis of Table 4.8, we can see that the original segmentation is better on the AxonEM-M dataset than on the SNEMI3D dataset, as we can see that the % of the ground truth ERL is higher on AxonEM-M. In addition, as explained in Section 3.5 when defining the ERL, this metric tends to penalize more merge errors. Despite the metric's greater penalty towards merge errors, the corrected segmentations, which may introduce some merge errors, still improve the results. This clearly indicates that the benefit of correcting split errors outweighs significantly any loss from new merge errors.

On the SNEMI3D test dataset, out of the 51 split errors present in the original segmentation, *PoinNet* [43] corrected 38 of them, and created 3 false merges. On the AxonEM dataset, out of the 89 split errors present in the original segmentation, the best model *2.5D CNN* corrected 65 of them and created 2 false merges. These results are promising, with most errors being corrected and only a few being introduced.

Illustration of corrected segmentation

Figure 4.7 illustrates the correction of segments of the original segmentation. We depict a case of split error corrected and merge error created on both datasets.



(a) SNEMI3D (b) AxonEM-M

Figure 4.7: **Illustration of corrected axons**. Left of (a): All the split errors are corrected. Right of (a): the split errors are corrected, but a merge error is added. Left of (b): the split error is corrected. Right of (b): a merge error is added.

Chapter 5

Discussion

5.1 Error analysis of the detection via skeletonization

As observed from the results of the detection in Section 4.1.2, the detection method exhibited a low precision and produced a significant number of false positives. This outcome can be attributed to the presence of "fake" endpoints on the skeleton, which do not correspond to genuine endpoints on the corresponding $3D$ structure. As a result, the candidates detected around these "fake" endpoints are more likely to be false positives since real split errors typically occur only at the endpoints of the $3D$ structure.

The existence of "fake endpoints" can be traced back to the noise present in the skeletons generated by the complex and anisotropic $3D$ structure. Notably, this issue of noisy endpoints was more pronounced on the SNEMI3D dataset, leading to a higher incidence of false positives on this data. Figure 5.1 illustrates axons with their corresponding skeletons, with each white dot depicting an endpoint on the skeleton.

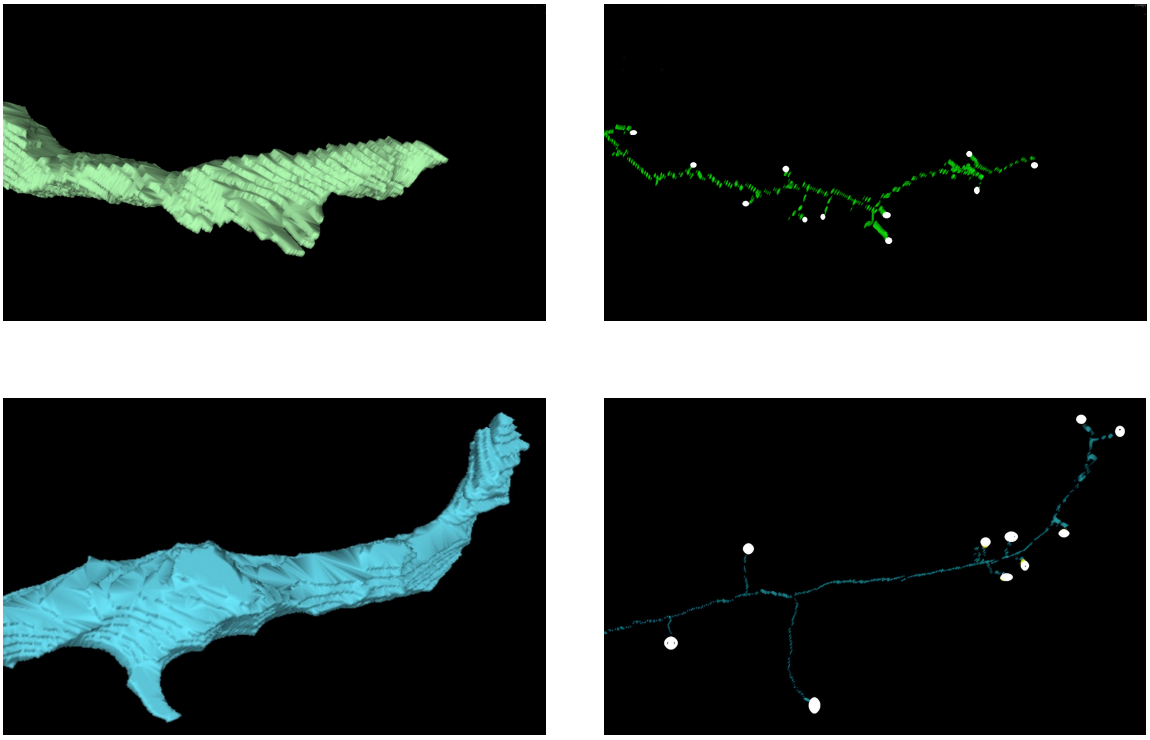


Figure 5.1: **Axons from the SNEMI3D dataset, with their corresponding noisy skeletons.** Left: Axons. Right: Corresponding skeletons. White dots are added in post-processing to depict the endpoints of the skeleton. The skeletons were computed with the skeletonize function from Scikit-image [44]. In an ideal scenario, the axons should have one or two endpoints on their skeletons, aligning with the tip of their $3D$ structure. However, approximately 10 endpoints are predicted, indicating a significant overestimation.

As described in Section 4.1.2, we also implemented the skeleton function from the Kimimaro package *et all* [48] on the SNEMI3D dataset but it resulted in a lower precision as it generated even more noisy endpoints. This outcome from both skeleton algorithms suggests that the issue is likely to be inherent to the $3D$ representations in the SNEMI3D dataset. We attempted to address this challenge by filtering out the endpoints that were not at the end of a long enough branch on the skeleton, given that most of the noisy endpoints were associated with a small branch. However, this approach proved impractical as it eliminated too many real candidates and added excessive computational complexity to the method. The AxonEM-M datasets had generally less noisy skeletons, which is reflected in the higher precision results presented in Section 4.1.2.

False positive candidates may also arise due to densely packed axons, where multiple real endpoints are present in the same region. Figure 5.2 highlights two types of false positives that can be generated. The first type, illustrated with the two images at the top of Figure 5.2, arises due to the proximity of two endpoints in space, even though it is not an actual error. The second type, depicted by the two images at the bottom of Figure

5.2, arises from noisy endpoints on the skeleton that do not correspond to an endpoint on the corresponding 3D structure.

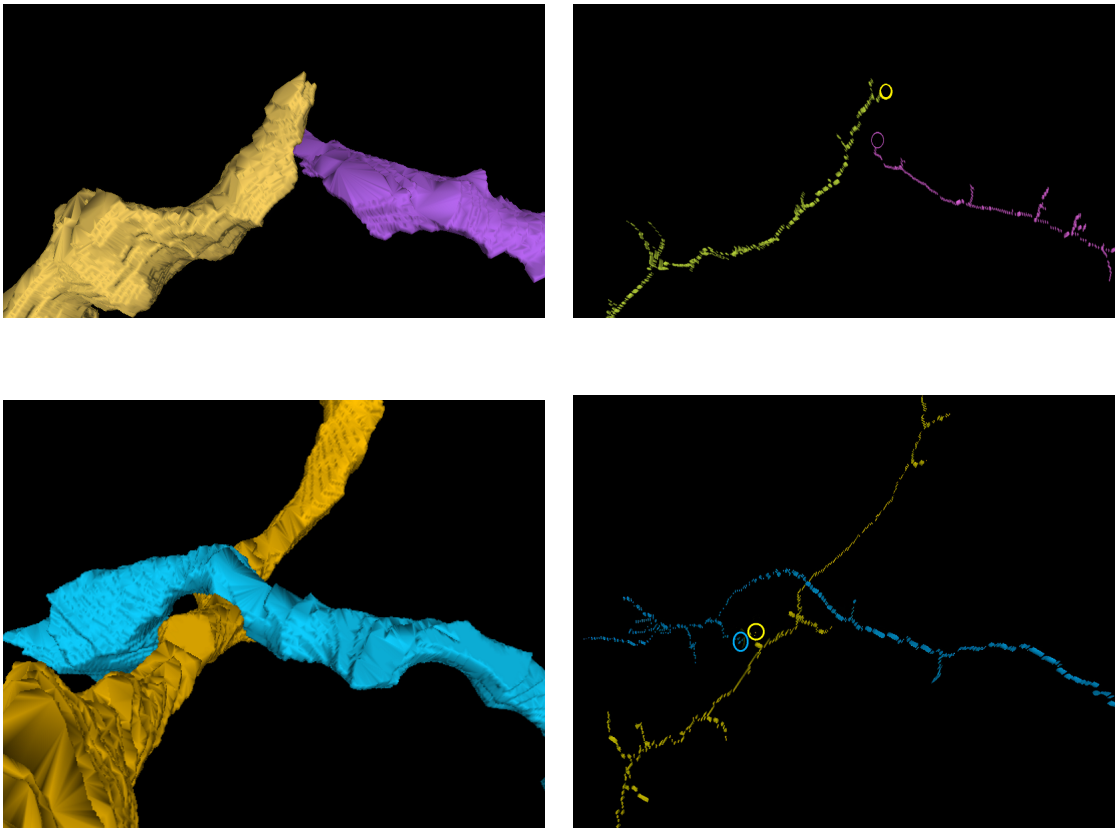
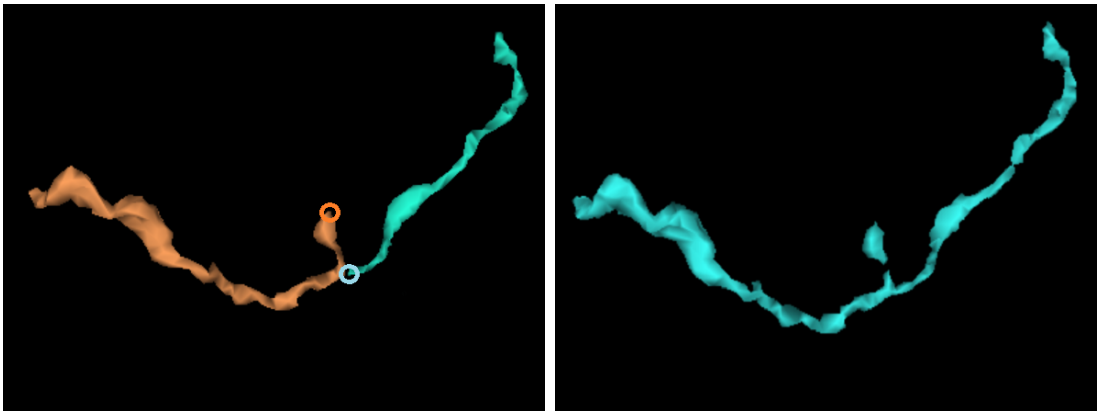


Figure 5.2: **Illustration of false positive candidates from the detection phase.** The circles on the skeletons indicate the pairs of endpoints. **Top:** False positive generated due to the proximity of two axon endpoints. **Bottom:** False positive caused by noisy endpoints. It can be observed that the endpoints identified are not genuine endpoints on the axons.

These visualizations are demonstrated using the SNEMI3D dataset since the skeleton algorithm from Skimage-image [44] directly outputs an image, whereas the Kimmimaro function [40] generates a graph that is not immediately usable for visualization. However, these phenomena happened in both datasets, but with a relatively lesser frequency in the AxonEM-M dataset, as the higher precision in Table 4.2 showed.

In contrast to the SNEMI3D dataset, where all the candidates were identified (recall = 1), some split errors were not detected on the AxonEM-M dataset. An example of a missed element is illustrated in Figure 5.3. The left of the figure illustrates the split error in the segmentation, with the circles indicating the two endpoints detected via skeletonization. The two endpoints were not close enough, resulting in their omission by Algorithm 1.



(a) Split error in the segmentation (b) Ground truth

Figure 5.3: **Illustration of a missed error in the detection phase on the AxonEM-M dataset.** The two endpoints depicted by the circles on the left were not close enough to be detected by Algorithm 1.

5.2 Error analysis on point cloud

Based on the results of Section 4.2.5 and the final impact on the segmentation presented in Section 4.3, *PointNet* [43] has emerged as one of the top-performing models. Given that axon segmentation typically entails voluminous data, the computational efficiency and memory usage advantages of the point cloud approach are highly valuable, rendering it the optimal choice for scaling. Therefore, we focused on analyzing the errors generated by the point cloud model.

To gain insights into the point cloud, we can examine the points activated by the max-pooling layer in *PointNet* [43]. It transforms a $1024 \times N$ matrix, where 1024 denotes the number of features and N denotes the number of points, into a 1024×1 vector. For each feature, the max-pool layer selects the point with the highest value concerning that feature. Points that are not activated do not impact the final prediction. Visualizing these points can aid in comprehending the model. Figure 5.4 illustrates the critical points in successful predictions for both datasets with *PointNet* [43]. It is apparent that the critical points are distributed evenly across both objects, which is logical. It makes sense that the model seems to weigh each object similarly. Furthermore, we can see that the critical points summarize the global shape of the input by capturing its overall structure.

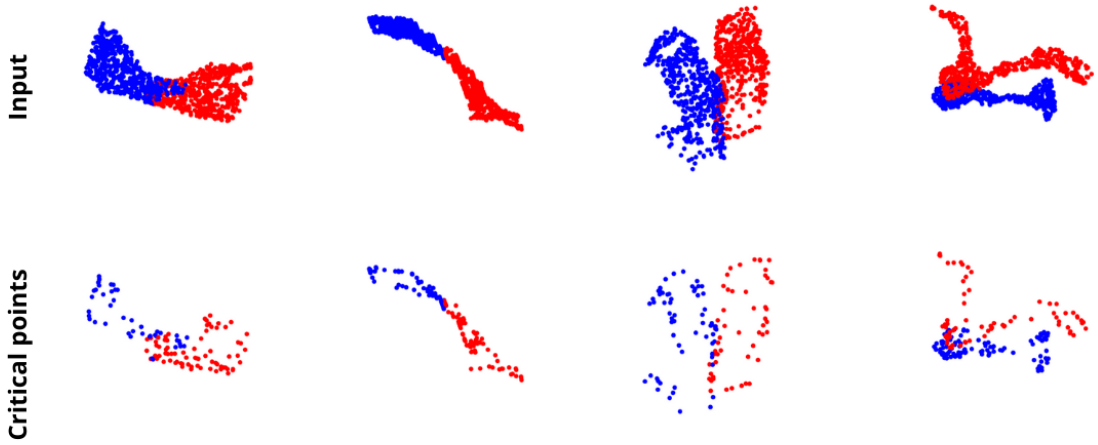


Figure 5.4: **Visualization of the critical points**. The image illustrates true positives and true negatives for both datasets. From left to right: TP from SNEMI3D, TP from AxonEM-M, TN from SNEMI3D, TN from AxonEM-M.

False positives

As we aimed to prioritize precision by employing a high threshold (0.81 and 0.72 for SNEMI3D and AxonEM-M, respectively) on the classifier prediction, the false positives identified by the models are indicative of significant prediction errors. One of the contributing factors to these errors is the small size of the training dataset, which may not be adequate for the model to accurately classify inputs that it has not encountered before. Figure 5.5 showcases examples of false positives on both datasets. We can see that in most cases, particularly in the first, second, and fourth columns, the input distribution is relatively similar to that of the positive samples. The objects are not entirely dissimilar but exhibit a shared locality, which could potentially trigger the model to make an error.

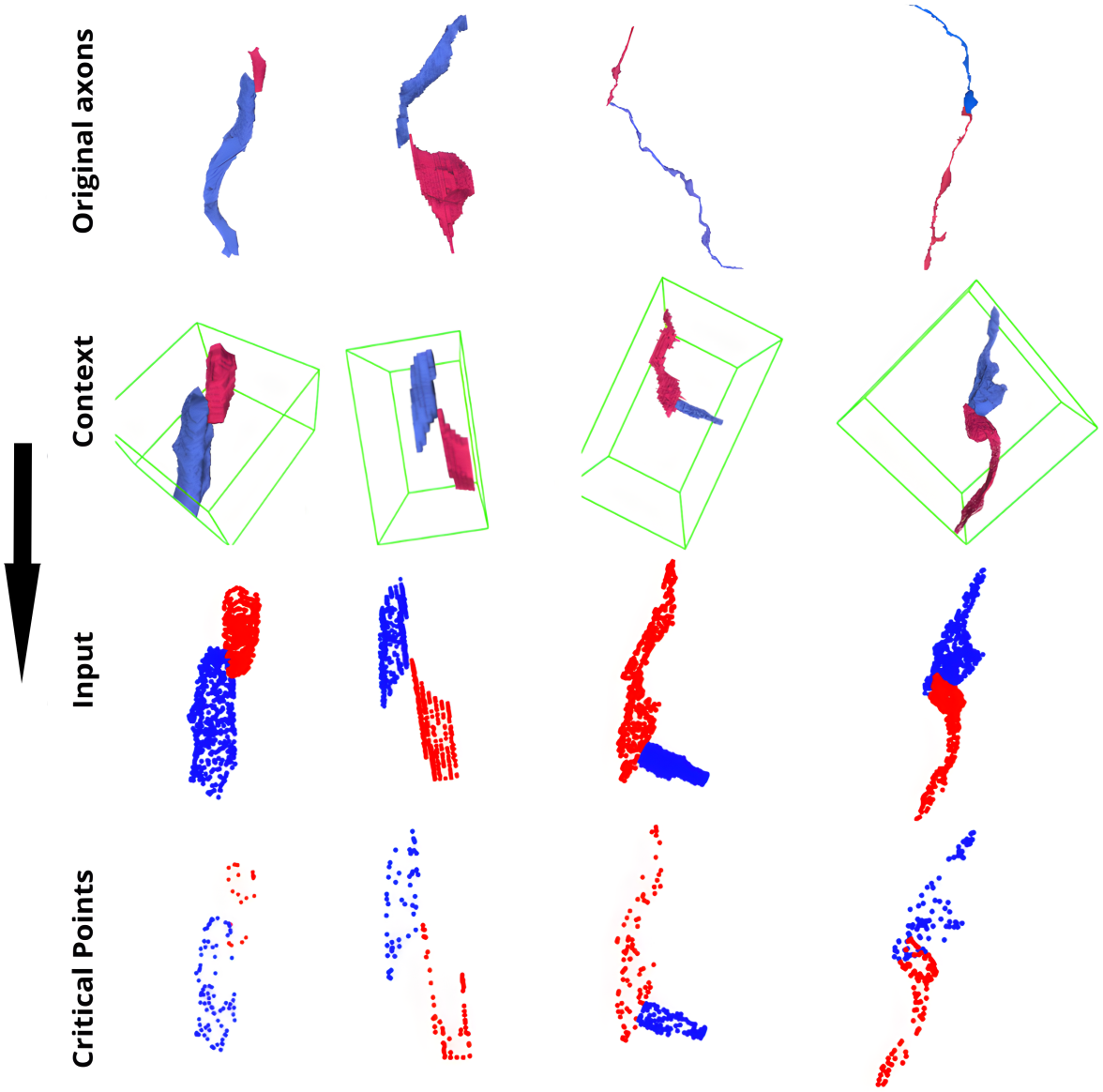


Figure 5.5: **Illustration of false positives.** The two left columns illustrate results from the SENMI3D dataset. The two right columns illustrate results from the AxonEM-M dataset. Model predictions of the input from left to right: 0.9121, 0.8853, 0.9218, 0.9289.

False negatives

Conversely, unlike the false negatives, many false positives can be attributed to the too high threshold on the classifier prediction. Figure 5.6 presents some wrong predictions. In the case of the SNEMI3D dataset, we observe that the incorrectly predicted objects do not resemble the typical positive samples shown earlier in Figure 5.4, but rather represent small encrusted parts. On the other hand, for the AxonEM-M dataset on the right of the figure, it appears that a high angle change at the intersection region might lead the model to make an erroneous prediction.

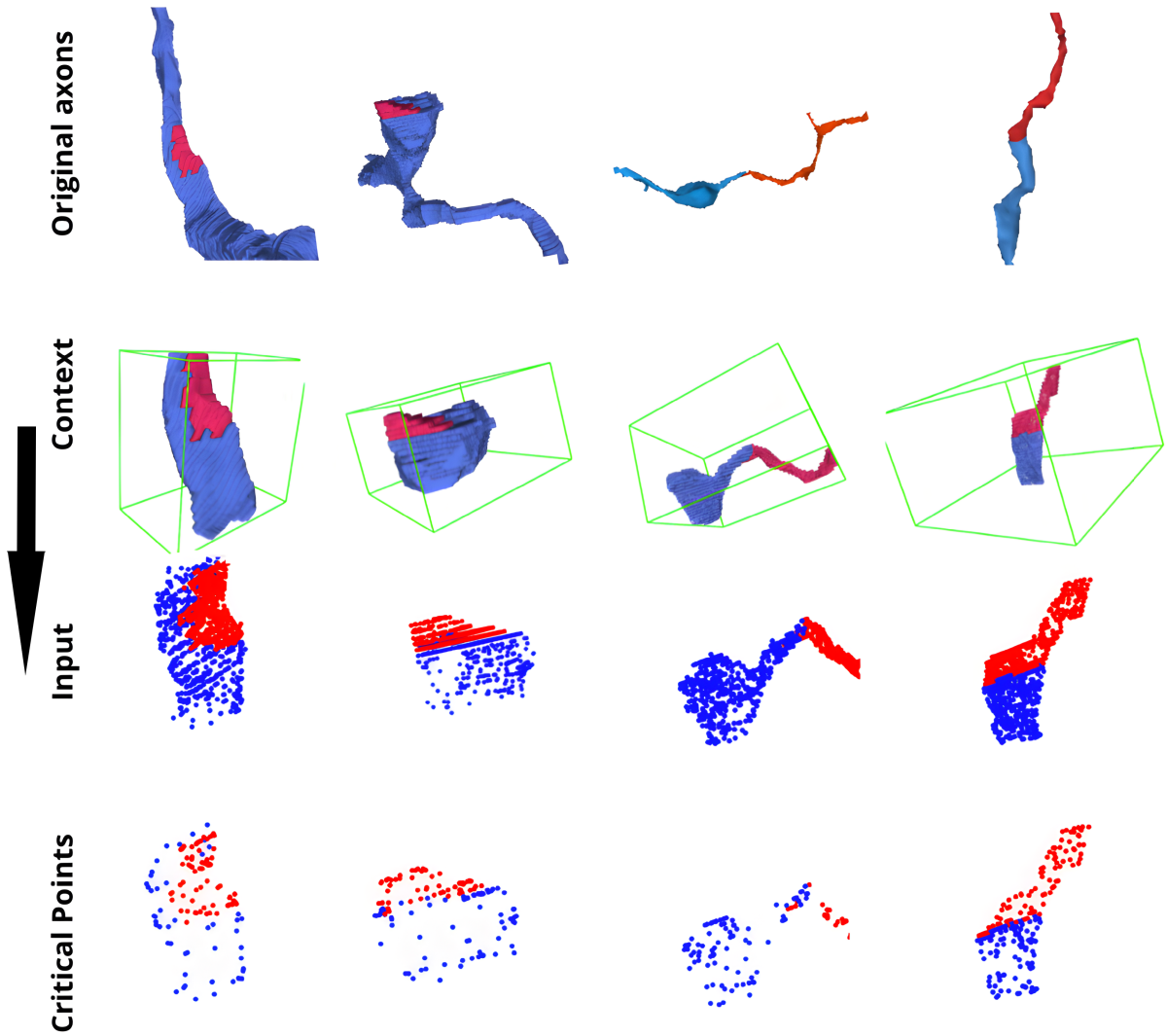


Figure 5.6: **Illustration of false negatives.** The two left columns illustrate results from the SENMI3D dataset. The two right columns illustrate results from the AxonEM-M dataset. Model predictions of the input from left to right: 0.2909, 0.1099, 0.29, 0.3875.

5.3 Limitations and future work

One important limitation is that a substantial amount of data for training (50%) is required for the classification. Manual labeling and proofreading of ground truth in Connectomics is extremely time-consuming due to the volumetric nature of the segmentations. Ideally, we would use fewer samples for training. To achieve this, one approach would be to create good synthetic 3D images, for example with 3D GANs [53–55] or diffusion models [56]. As explained in Section 3.3.1 and as described in Appendix A, we explored the creation of synthetic images but our approach was too simple. Another approach would be to augment the data by proposing similar objects to the training data in an unlabeled dataset. This could be seen as an object proposal problem, generating a set

of candidates having similar morphology [58]. It could also be seen as a semi-supervised learning, where humans would validate some of the proposed input. Another approach would be to incorporate the ideas developed in Sun, Deqing, *et al* [57], where they propose a method to generate augmented training data for optical flow estimation using a differentiable data augmentation pipeline. Their techniques of framing the generation of training data as a unified optimization problem that combines the rendering of data with the training of the model could be useful to build more sophisticated training data.

Another limitation of the method is that it only corrects split errors. It would be interesting to explore the correction of merge errors too. The challenge of addressing merge errors lies in identifying the precise location where two objects were merged, making it a difficult problem to solve. Some work, such as Konstantin Dmitriev *et al.* [38], have explored the use of linear cut planes to correct false merges in neuron segmentation.

5.4 Conclusion

This work presents a comprehensive analysis of split error correction in axon segmentation. The corrected segmentation results demonstrate significant improvement over the initial segmentation on both datasets. This improvement is evidenced by the increase in three distinct metrics, providing valuable evaluation from different perspectives on the performance of the correction method. These findings confirm the effectiveness of the correction method for axon segmentation. In addition, we demonstrate that the point cloud approach achieves some of the best results while being computationally less expensive and memory-efficient. This is particularly important in the context of connectomics where data are highly volumetric, often reaching terabytes. Through our analysis, we discovered that the point cloud models were able to accurately understand the underlying structure of axons, highlighting critical points that summarize their morphologies effectively. Furthermore, our investigation into the skeleton method for detection identified the primary source of errors as being caused by noisy skeletons resulting from the complex 3D structures and anisotropic nature of the data.

Bibliography

[1] Zlateski, Aleksandar, and H. Sebastian Seung. "Image segmentation by size-dependent single linkage clustering of a watershed basin graph." arXiv preprint arXiv:1505.00249 (2015).

[2] Funke, Jan, et al. "Large scale image segmentation with structured loss based deep learning for connectome reconstruction." IEEE transactions on pattern analysis and machine intelligence 41.7 (2018): 1669-1680.

[3] Lee, Kisuk, et al. "Superhuman accuracy on the SNEMI3D connectomics challenge." arXiv preprint arXiv:1706.00120 (2017)

[4] Zlateski, Aleksandar, and H. Sebastian Seung. "Image segmentation by size-dependent single linkage clustering of a watershed basin graph." arXiv preprint arXiv:1505.00249 (2015).

[5] Cousty, Jean, et al. "Watershed cuts: Minimum spanning forests and the drop of water principle." IEEE transactions on pattern analysis and machine intelligence 31.8 (2008): 1362-1374.

[6] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18. Springer International Publishing, 2015.

[7] Ciresan, Dan, et al. "Deep neural networks segment neuronal membranes in electron microscopy images." Advances in neural information processing systems 25 (2012).

[8] Knowles-Barley, Seymour, et al. "RhoanaNet pipeline: Dense automatic neural annotation." arXiv preprint arXiv:1611.06973 (2016).

[9] Andres, Bjoern, et al. "Globally optimal closed-surface segmentation for connectomics." Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part III 12. Springer Berlin Heidelberg, 2012.

[10] Nunez-Iglesias, Juan, et al. “Machine learning of hierarchical clustering to segment 2D and 3D images.” PloS one 8.8 (2013): e71715.

[11] Parag, Toufiq, et al. “Anisotropic EM segmentation by 3D affinity learning and agglomeration.” arXiv preprint arXiv:1707.08935 (2017).

[12] Meirovitch, Yaron, et al. “A multi-pass approach to large-scale connectomics.” arXiv preprint arXiv:1612.02120 (2016).

[13] Januszewski, Michał, et al. “High-precision automated reconstruction of neurons with flood-filling networks.” Nature methods 15.8 (2018): 605-610.

[14] Gonda, Felix, Donglai Wei, and Hanspeter Pfister. “Consistent recurrent neural networks for 3d neuron segmentation.” 2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI). IEEE, 2021.

[15] Shi, Xingjian, et al. “Convolutional LSTM network: A machine learning approach for precipitation nowcasting.” Advances in neural information processing systems 28 (2015).

[16] Cuisenaire, Olivier, et al. “Automatic segmentation and measurement of axons in microscopic images.” Medical Imaging 1999: Image Processing. Vol. 3661. SPIE, 1999.

[17] Yu, Yong, et al. ”A review of recurrent neural networks: LSTM cells and network architectures.” Neural computation 31.7 (2019): 1235-1270.

[18] More, Heather L., et al. “A semi-automated method for identifying and measuring myelinated nerve fibers in scanning electron microscope images.” Journal of neuroscience methods 201.1 (2011): 149-158.

[19] Wang, Yi-Ying, et al. “Segmentation of nerve fibers using multi-level gradient watershed and fuzzy systems.” Artificial intelligence in medicine 54.3 (2012): 189-200.

[20] Zaimi, Aldo, et al. “AxonSeg: open source software for axon and myelin segmentation and morphometric analysis.” Frontiers in neuroinformatics 10 (2016): 37

[21] Mesbah, Rassoul, Brendan McCane, and Steven Mills. “Deep convolutional encoder-decoder for myelin and axon segmentation.” 2016 International Conference on Image and Vision Computing New Zealand (IVCNZ). IEEE, 2016.

[22] Zaimi, Aldo, et al.“AxonDeepSeg: automatic axon and myelin segmentation from microscopy data using convolutional neural networks.” Scientific reports 8.1 (2018): 3816.

[23] Lee, Kisuk, et al. “Convolutional nets for reconstructing neural circuits from brain images acquired by serial section electron microscopy.” Current opinion in neurobiology 55 (2019): 188-198.

[24] Wei, Donglai, et al. “AxonEM dataset: 3d axon instance segmentation of brain cortical regions.” Medical Image Computing and Computer Assisted Intervention–MICCAI 2021: 24th International Conference, Strasbourg, France, September 27–October 1, 2021, Proceedings, Part I 24. Springer International Publishing, 2021.

[25] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation.” Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18. Springer International Publishing, 2015.

[26] Ciresan, Dan, et al. “Deep neural networks segment neuronal membranes in electron microscopy images.” Advances in neural information processing systems 25 (2012).

[27] Januszewski, Michał, et al. “High-precision automated reconstruction of neurons with flood-filling networks.” Nature methods 15.8 (2018): 605-610.

[28] Naito, Tatsuhiko, et al. “Identification and segmentation of myelinated nerve fibers in a cross-sectional optical microscopic image using a deep learning model.” Journal of neuroscience methods 291 (2017): 141-149.

[29] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation.” Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18. Springer International Publishing, 2015.

[30] Motta, Alessandro, et al. “Dense connectomic reconstruction in layer 4 of the somatosensory cortex.” Science 366.6469 (2019): eaay3134.

[31] More, Heather L., et al. “A semi-automated method for identifying and measuring myelinated nerve fibers in scanning electron microscope images.” Journal of neuroscience methods 201.1 (2011): 149-158.

[32] Wang, Yi-Ying, et al. “Segmentation of nerve fibers using multi-level gradient watershed and fuzzy systems.” Artificial intelligence in medicine 54.3 (2012): 189-200.

[33] Haehn, Daniel, et al. “Design and evaluation of interactive proofreading tools for connectomics.” IEEE transactions on visualization and computer graphics 20.12 (2014): 2466-2475.

[34] Knowles-Barley, Seymour, et al. “Mojo 2.0: Connectome annotation tool.” Frontiers in Neuroinformatics 60 (2013): 1.

[35] Haehn, Daniel, et al. “Guided proofreading of automatic segmentations for connectomics.” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018.

[36] Matejek, Brian, et al. “Biologically-constrained graphs for global connectomics reconstruction.” Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019.

[37] Berman, Jules, Dmitri B. Chklovskii, and Jingpeng Wu. “Bridging the Gap: Point Clouds for Merging Neurons in Connectomics.” International Conference on Medical Imaging with Deep Learning. PMLR, 2022.

[38] Dmitriev12, Konstantin, et al. “Efficient correction for em connectomics with skeletal representation.” British Machine Vision Conferemce (BMVC) (2018).

[39] Zung, Jonathan, et al. “An error detection and correction framework for connectomics.” Advances in neural information processing systems 30 (2017).

[40] W. Silversmith, J.Alexander Bae, Peter H. Li, A.M. Wilson, “Kimimaro: Skeletonize densely labeled 3D image segmentations” (2021). Code available on `https://github.com/seung-lab/kimimaro` (visited on 03/02/2022)

[41] Qishen Ha, winner of the RSNA 2022 Cervical Spine Fracture Detection Kaggle competition (2022). Retrieved from `https://www.kaggle.com/competitions/rsna-2022-cervical-spine-fracture-detection/discussion/362607` (visited on 03/02/2023)

[42] Keuper, Margret, et al. “Efficient decomposition of image and mesh graphs by lifted multicuts.” Proceedings of the IEEE international conference on computer vision. 2015.

[43] Qi, Charles R., et al. “Pointnet: Deep learning on point sets for 3d classification and segmentation.” Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[44] Van der Walt, Stefan, et al. “scikit-image: image processing in Python.” PeerJ 2 (2014): e453.

[45] Zhang, Tongjie Y., and Ching Y. Suen. “A fast parallel algorithm for thinning digital patterns.” Communications of the ACM 27.3 (1984): 236-239.

[46] Sato, Mie, et al. “TEASAR: tree-structure extraction algorithm for accurate and robust skeletons.” Proceedings the Eighth Pacific Conference on Computer Graphics and Applications. IEEE, 2000.

[47] Smith, Leslie N. “Cyclical learning rates for training neural networks.” 2017 IEEE winter conference on applications of computer vision (WACV). IEEE, 2017.

[48] Silversmith, W., et al. “Kimimaro: Skeletonize densely labeled 3D image segmentations.” Zenodo https://doi. org/10.5281/zenodo 5539912 (2021).

[49] Kingma, Diederik P., and Jimmy Ba. “Adam: A method for stochastic optimization.” arXiv preprint arXiv:1412.6980 (2014).

[50] Ilya, Loshchilov, and Hutter Frank. “Decoupled weight decay regularization.” Proceedings of ICLR 7 (2019).

[51] Liu, Ting, et al. “A modular hierarchical approach to 3D electron microscopy image segmentation.” Journal of neuroscience methods 226 (2014): 88-102.

[52] Meilă, Marina. “Comparing clusterings by the variation of information.” Learning Theory and Kernel Machines: 16th Annual Conference on Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003. Proceedings. Springer Berlin Heidelberg, 2003.

[53] Chan, Eric R., et al. “pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis.” Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2021.

[54] Han, Changhee, et al. “Synthesizing diverse lung nodules wherever massively: 3D multi-conditional GAN-based CT image augmentation for object detection.” 2019 International Conference on 3D Vision (3DV). IEEE, 2019.

[55] Sun, Li, et al. “Hierarchical amortized gan for 3D high resolution medical image synthesis.” IEEE journal of biomedical and health informatics 26.8 (2022): 3966-3975.

[56] Watson, Daniel, et al. “Novel view synthesis with diffusion models.” arXiv preprint arXiv:2210.04628 (2022).

[57] Sun, Deqing, et al. “Autoflow: Learning a better training set for optical flow.” Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021.

[58] Chen, Xiaozhi, et al. “3d object proposals for accurate object class detection.” Advances in neural information processing systems 28 (2015)

Appendix

A Creation of synthetic images

The binary classification task on the SNEMI3D dataset is highly unbalanced, with around 85% of the examples labeled as negative. To address this issue and improve the performance of the models, we explored the creation of synthetic positive images for the training. One approach to generating synthetic split errors is to create a false split in an axon by artificially cutting it into two pieces. To explore this idea, we implemented an algorithm to perform the false split by cutting the axon perpendicular to its main direction, as illustrated in Figure 7. The algorithm that we used to perform the cut is described in Algorithm 2:

Algorithm 2 Synthetic image creation

Choose an axon.
Compute a cube around one part of the axon.
Compute the vector between two extreme points of the axon.
Compute the equation of the plan centered in the middle of the axon.
Cut the axon with this plan, i.e voxels below the plan correspond to the first segment, and above correspond to the second segment.

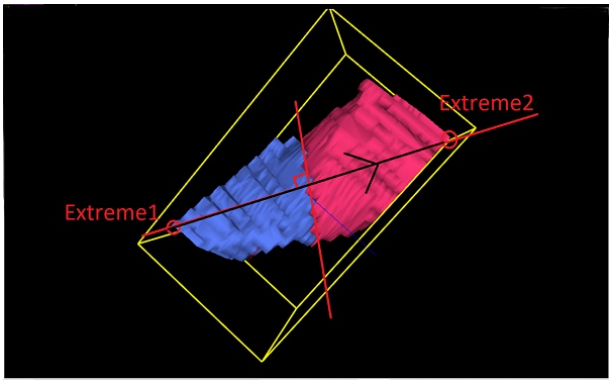


Figure 7: **Illustration of a synthetic image.** The black vector indicates the direction between the two extreme points of the global structure, and the red line, perpendicular to the black vector, indicates the cutting plan.

The corresponding images did not improve the performance of the models. The synthetic images are too simple with only a linear boundary.

B Source code

All the code was written in Python. Source code can be found at `https://github.com/AlexandreDiPiazza/Master_Thesis`