*Department of Construction Sciences*
Solid Mechanics

ISRN LUTFD2/TFHF-23/5254-SE(1-25)

# Identification of Fibers in Micro-CT Images of Paperboard Using Deep Learning

Master's Dissertation by

David Rydgård

Supervisors:
Sara Johansson, PhD. Division of Solid Mechanics
Johan Tryding, Adj. Prof. Division of Solid Mechanics and Tetra Pak

Examiner:
Stephen Hall, Assoc. Prof. Division of Solid Mechanics

## Abstract

This master thesis project explores the possibility of using deep learning to segment individual fibers in three-dimensional tomography images of paperboard fiber networks. We test a method which has previously been used to segment fibers in images of glass fiber reinforced polymers. The method relies on a neural network which produces an embedding for each voxel in the input image, such that the embeddings corresponding to a given fiber should form a cluster in the embedding space. Individual fibers can then be identified by applying a clustering algorithm to the embeddings. Although the method is able to identify some more easily distinguished fibers, the achieved accuracy is insufficient. We find that the main difficulty lies in acquiring training data of high enough quality, and that future work concerning this task is required. In this work, the use of different types of data, including synthetically generated data, and what we refer to as a type of semi-synthetic data, have been tested. Although we do not reach any satisfying results, this work may serve as a base for future research.

# Preface

This master thesis project has been carried out at the division of Solid Mechanics, Lund University, in collaboration with Tetra Pak, with the help of my supervisors Sara Johansson and Johan Tryding. It has been a challenging endeavor, accompanied by hope and optimism which could be accurately modeled using a sinusoidal. Although high ambitions have not been met, I am happy to say that I have had the opportunity to work on an interesting and engaging task, and that I have learned a lot along the way.

This project would not have been possible without my supervisors, Sara and Johan. I am grateful for their patience and the support that they have provided throughout this work. I hope that I have been able to contribute with some insight into the application of machine learning to the problem of identifying fibers - if not regarding possibilities, at least regarding limitations.

# Contents

# 1 Introduction

This project has been carried out at the division of Solid Mechanics, Lund University, in collaboration with Tetra Pak. Tetra Pak is the world's largest producer of paperboard based packages, with hundreds of billions of packages produced every year. A good understanding of paper properties is important for the packaging industry, as it allows for using raw material more efficiently, producing more durable packages, etc. The properties of paper, both mechanical and optical, depend on the internal structure of the material. Tensile strength and fracture toughness are two examples of important properties of packaging materials.

With few exceptions, the raw material used for producing paper is wood. The wood is chopped into small wood chips, and these are converted to free fibers. A suspension of water and wood fibers is sent into a paper machine, where it is turned into paper through several steps of forming and removing of water. The structure and properties of the resulting fiber network follow from the pulp used, and different factors in the paper machine operations.[1]

The essential characteristics of paper are the fiber structural properties and network connectivity[2]. Being able to model the fiber networks it is essential to have reliable estimations of these characteristics. In one way or another, all mechanical properties are controlled by the inter-fiber bonds in the fiber network. Typically, there are between 10 and 40 such bonds per fiber[3]. This number can influence both the stiffness and tensile strength.

X-ray micro-CT imaging has become a valuable and widely used tool for the analysis of structural materials[4]. By identifying and isolating each individual fiber in the 3D network, it is possible to get access to fiber properties such as length, cross-section area and orientation, as well as certain network level properties such as the number of bonds between fibers. Doing this manually is a very time-consuming task, and it would be valuable if this identification of fibers could be done automatically.

There have been attempts to identify each individual fiber using traditional image analysis methods. For example, Sharma et al. (2015)[5] presented a method based on tracking the hollow part of each fiber, called the lumen, using a modified connected components algorithm. Aronsson (2002)[1] used a combination of two methods developed for segmentation of arteries in medical images; Ordered Region-Growing and SeparaSeed. However, no method has been able to accurately segment a large fraction of the fibers in an image.

Deep learning has previously been used for segmentation of fiber-reinforced composites[6][7]. In paper, the fiber density is higher, and there is larger variation in shape, size and orientation of fibers, making the segmentation of paper fibers a more challenging problem. However, due to its successful application to similar problems, it seems that deep learning has the potential of segmenting fibers with higher accuracy than what has been achieved using traditional image analysis methods. The aim of this project has been to examine the possibility of using deep learning to identify individual fibers in tomography images of paper fiber networks.

### 1.1 Image Segmentation

Image segmentation is a subdomain of image analysis, consisting of methods for partitioning an image into different segments, where each segment represents a certain category or object. There are two main types of image segmentation; semantic segmentation and instance segmentation. In semantic segmentation, each pixel is assigned a label representing a certain category of objects it belongs to. E.g., in the case of fiber networks, the goal of semantic segmentation can be to label a pixel as a 'fiber pixel', or as a 'background pixel'. In instance segmentation, the goal is to identify which category a pixel belongs to, and also differentiate between different objects, or *instances*, in that category.

In this project the goal has been to perform instance segmentation on tomograms of fiber networks, i.e. to label each voxel depending on whether it is a fiber voxel, and also depending on which fiber it is part of. Voxels belonging to the same fiber should be given the same label, and voxels belonging to different fibers should be given different labels.

# 2 Theoretical Background

## 2.1 Artificial Neural Networks

An artifical neural network (ANN) is a type of machine learning model which takes an input, and produces an output. Typically, an ANN consists of layers of nodes, successively transforming the input data. Figure 1 shows an illustration of a simple network, which takes four values as input and produces two output values. Each node weights and sums the outputs from the previous layer, adds a bias term, and applies a nonlinear function $f$, often referred to as the *activation function.* I.e., the output $y$ from a node is calculated as

$$y = f(\boldsymbol{w}^T \boldsymbol{x} + b), \tag{1}$$

where $\boldsymbol{x}$ is the input to the node, $\boldsymbol{w}$ is the weight vector, $b$ is the bias, and $f$ is the activation function. One commonly used function is the *rectified linear unit* (ReLU):

$$f(x) = \max(0, x)$$

The activation function in the output layer defines the types of predictions the model can be used for. For example, a neural network that is used for binary classification often has an output layer with the *sigmoid* activation function:

$$f(x) = \frac{1}{1 + e^{-x}}$$

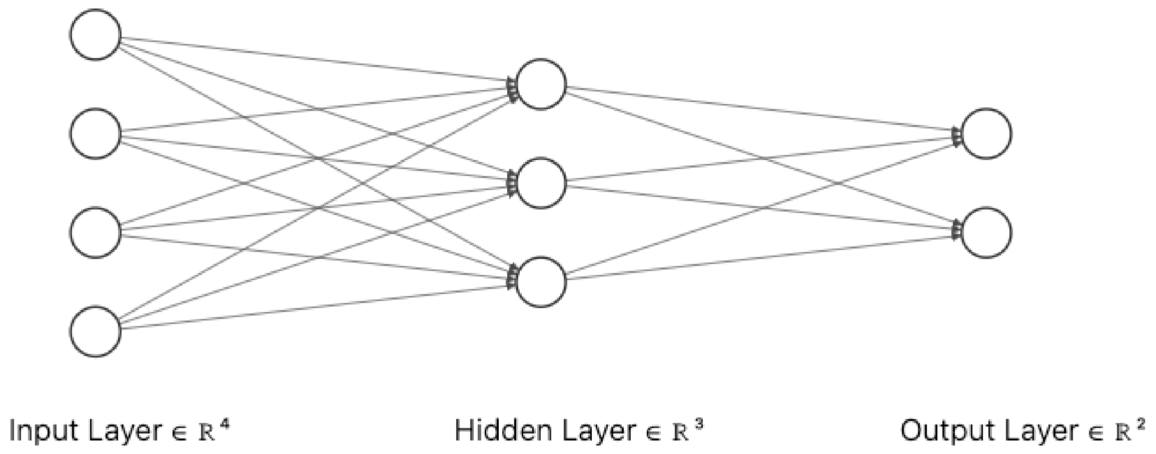


Figure 1: Illustration of how data propagates through a simple ANN with four input nodes, one so-called *hidden layer* with three nodes, and two output nodes. Each node has its own weight vector and bias, and performs the calculation in eq. (1).

This function gives an output value between 0 and 1, which can then be compared to a classification threshold.

### 2.1.1 Training

When used to make inference, a neural network takes an input $X$ and makes a prediction of a variable $Y$. To be able to make accurate predictions, the network first has to be trained on a training data set. Such a data set consists of a number of data points with inputs $X_i$ and corresponding *target values* $Y_i$. The model is trained by adjusting the values of the parameters in the network so that it performs well on the training data, i.e. by fitting the model to the training data.

More precisely, the inputs are run through the network, and the resulting outputs are passed to a *loss function*, which is a function that takes on a value based on how close the outputs are to the desired values. Generally, a loss function takes on lower values for more desirable outputs. This means that the network parameters can be adjusted using gradient descent, or a variation thereof. The described procedure is repeated for a, typically large, number of iterations.

It is possible to *overfit* a model to the training data, meaning that the model starts to fit to the inherent noise, or randomness, in the training data. This can negatively impact the *generalization*, i.e. the ability of the model to perform well on new, unseen data. In order to observe how the generalization is affected during training, a set of *validation data* can be used. The model is not trained on this data, i.e. the data does not affect the updates of the network weights, but it can be used to calculate some performance metric that gives an indication of how the generalization changes during training.

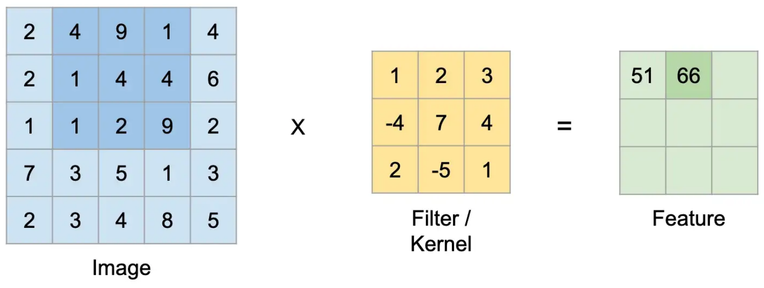


Figure 2: Illustration of the operation performed by a 2D convolutional layer that takes a single-channel input, and has a 3x3 kernel. Each 3x3 patch in the input is multiplied element-wise by the convolution kernel, and the resulting values are summed up, yielding the value in the corresponding position in the output feature map. E.g. the highlighted value in the image is calculated as $1 \cdot 4 + 2 \cdot 9 + 3 \cdot 1 - 4 \cdot 1 + 7 \cdot 4 + 4 \cdot 4 + 2 \cdot 1 - 5 \cdot 2 + 1 \cdot 9 = 66$. The image is taken from [8].

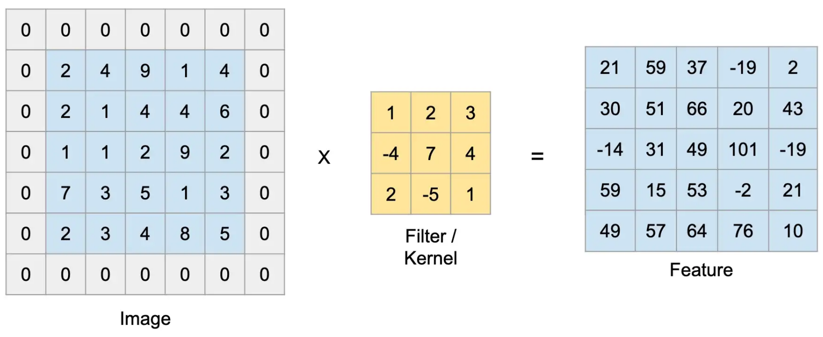


Figure 3: The same operation as in figure 2, but with the input padded with zeros. The padding leads to the output feature map having the same height and width as the input. Image taken from [8].

### 2.1.2 Convolutional Neural Networks

A convolutional neural network (CNN) is a neural network that contains at least one *convolutional layer*. In the last decade, CNNs have come to be widely used in the field of image analysis[9]. They are mostly used for dealing with 2D-images, but can also be used with 3D-images. In the 2-dimensional case, a convolutional layer takes an input of dimensions $C$x$H$x$W$. If the input is an image, $C$ is the number of channels, and $H$ and $W$ are the height and width, respectively.

The operation that the layer performs can be viewed as sliding a filter, i.e. a tensor of dimensions $C$x$S$x$S$, along the width- and height dimensions of the input, and at each position calculating the dot product of the filter and the corresponding patch of the input. The filter is often referred to as the *kernel*. Figure 2 illustrates the operation, for the case of a single-channel input and a 3x3 kernel. Here, each 3x3 patch in the input is multiplied element-wise by the kernel and the resulting values are summed up, yielding the value in the corresponding position in the output feature map. I.e. each value in the output is a weighted sum of the values in a patch of the input, with the weights being defined by the kernel. The operation of calculating this weighted sum is called convolution.

By applying the same calculations using a number $C_{out}$ of different kernels, an output with $C_{out}$ channels is obtained. Each channel contains what is often referred to as a *feature map*. A feature map can be thought of as a one-channel image, obtained by filtering the input image, using the corresponding kernel. This image contains representations of different features that the filtering has extracted from the input. When a CNN is trained, the kernel values are tuned such that the kernels are able to extract meaningful features.

The feature maps obtained from the above procedure are of smaller size than the input. To prevent this, the input can be padded with values along the edges, before the calculations. The values to use for padding can be chosen in different ways, but the most common method is to simply pad with zeros, as illustrated in figure 3.

#### 2.1.3 Hardware

The training of a neural network is generally a computationally demanding task, and often has rather high memory requirements. Since the performed calculations are conceptually simple to parallelize, the training process can be sped up significantly by using a GPU. Compared to a CPU, a GPU contains a large number of (slow) cores, making it efficient at performing a large number of simple computations, such as those involved in training a neural network, in parallel. However, this comes at a cost, as the amount of video random access memory (VRAM) in a GPU is generally more limited than the amount of RAM available to a CPU. This is of relevance in the memory-demanding case of dealing with 3D-images.

It is possible to use more than one GPU simultaneously. One way of doing this is using *data parallelism*. This means that the data in each minibatch, and the corresponding calculations, are distributed across multiple GPUs.

## 2.2 DBSCAN

Density-Based Spatial Clustering of Applications with Noise, or DBSCAN, is, as the name suggests, a density-based clustering algorithm. This means that it identifies areas of high density, i.e. areas containing many neighboring data points, in the data space. It depends on two parameters, $\varepsilon$ and *minSamples*. $\varepsilon$ defines the maximum distance, usually the Euclidean one, between two samples for the samples to be considered neighbors. Samples with at least *minSamples* neighbors are considered *core samples*. All samples (including ones that are core samples themselves) that are neighbors to a given core sample are labeled as belonging to the same cluster. If a sample is a neighbor of two core samples that belong to different clusters, the cluster it will be assigned to will depend on the order in which the data is provided to the algorithm. Samples that are not neighbors to any core sample (and are not core samples themselves) are considered outliers.

# 3 Related Work

## 3.1 Deep Learning-Based Methods

Using ANNs for instance segmentation is not as straightforward as for semantic segmentation. The semantic segmentation problem can be seen as a classification problem for each pixel, or voxel, with a predefined number of possible classes. Instance segmentation, on the other hand, can be viewed as a classification problem where the number of classes is not known beforehand, as each instance can be considered its own class. In a litterature review, two methods relevant to this project could be found; Mask R-CNN and the method presented in [10].

### 3.1.1 Mask R-CNN

The most well-known architecture for instance segmentation is Mask R-CNN, introduced by He et al. (2017)[11]. Mask R-CNN consists of a few stages. The first stage is a backbone which extracts feature maps from the input image. The second stage is known as a Region Proposal Network (RPN). The Region Proposal Network analyses the extracted features for each of a large number of predefined rectangular regions in the image; the implementer selects a few different aspect ratios and sizes for the rectangles, and then a large number of such rectangles, evenly spread out across the image, are analysed. For each region, the RPN attempts to determine the likelihood that the region contains an object. The regions that are the most likely to contain one object are passed along to the subsequent stages, where the objects are classified (possibly as 'background', if the region proposed by the RPN did not actually contain an object), and a *mask* for each object is generated. A mask is a binary image, with all pixels belonging to the object in question having the value '1', and all other pixels having the value '0'.

However, Mask-RCNN does not work very well with fibers, or elongated objects in general. This is due to the method being region-based. In a quadratic region covering a fiber, the fiber contributes very little information. The features that can be extracted from such a region are not informative enough for reliably segmenting the fiber.[12]

### 3.1.2 Approach Based on Clustering

In 2018, Konopczyński et al.[10] presented an approach for instance segmentation of fibers in low resolution 3D X-ray computed tomography scans of short glass fiber reinforced polymers. Below, this approach will be referred to as *Approach Based* *on Clustering* (ABC). ABC relies on a CNN with two separate branches. The two branches are independent of each other which means that they can be viewed as two different networks entirely, as will be the case in the following.

One of the networks performs semantic segmentation, and the other network produces an *embedding* for each voxel in the image. An embedding is a representation, in this case of a voxel, in the form of a vector. Konopczyński et al. used embeddings of length 16. The CNN is trained such that embeddings corresponding to the same fiber should form a cluster in the embedding space. To segment an image, a clustering algorithm (DBSCAN) is applied to the embeddings corresponding to those voxels which the semantic segmentation has identified as being 'fiber voxels'.

The network architectures are illustrated in figure 4, and will be referred to as the *semantic segmentation network* and *embeddings network*, respectively. Both architectures consist of a chain of residual blocks, followed by an output layer. A residual block, introduced by [13], consists of one or a few nonlinear layers and a skip connection, meaning that the input to the nonlinear layers is simply added to the output. All convolutions in the networks use kernels of size $C$x3x3, where $C$ is the number of channels in the input to the convolution, and padding of 1 to maintain the image size.

Figure 5 illustrates the residual block that we have used in this work. In this block, the convolution operation is applied twice. As in figure 4, the number of input and output channels, i.e. feature maps, are denoted $i$ and $o$, respectively. The first convolution operation is followed by the rectified linear unit. Since the number of channels is not the same after the first convolution, the skip-connection must include a (linear) transform. This transform ensures that the operands in the addition following the second convolution are of the same dimensions, so that the addition is possible. The addition is followed by batch normalization, which normalizes the data as described in [14], before the rectified linear unit is applied.

The output layer in the semantic segmentation network is a convolution layer which produces a one-channel output. The sigmoid activation function is used to scale the output values to the interval (0, 1). To classify each voxel as a 'fiber voxel' or 'background voxel', the network output is compared to a threshold of 0.5. The only difference between the architectures of the two networks is found in the output layers, as the embeddings network yields a 16-channel output and has no activation function.

Konopczyński et al. refer to the chains of residual blocks as the *backbones* of the networks. As can be seen in figure 4, these are identical in both architectures. Due to instability in the training of the embeddings network, Konopczyński et al. first trained the semantic segmentation network, and initialized the embeddings network backbone with the resulting backbone weights.

The semantic segmentation network is trained using voxel-wise cross-entropy error. With a total of $N_v$ voxels in a minibatch, and numbering these voxels as $1, ..., N_v$, the loss is defined as

$$L_{CE} = \frac{1}{N_v} \sum_{i=1}^{N_v} y_i \log \hat{y_i} - (1 - y_i) \log (1 - \hat{y_i}),$$

where $\hat{y_i}$ is the network output for voxel $i$. The ground truth $y_i$ is 1 if voxel $i$ is a fiber voxel, and 0 otherwise.

The embeddings network is trained using the loss function described in [10]. The loss for a single image is a weighted sum of three terms: $L_v$, $L_d$ and $L_r$. As explained by the authors, "$L_v$ keeps voxels belonging to the same object close to each other, $L_d$ (...) forces a minimal distance between clusters of different objects, and $L_r$ (...) regularizes the cluster centers to be close to the origin". The loss terms are defined as

$$L_v = \frac{1}{C} \sum_{c=1}^{C} \frac{1}{N_c} \sum_{i=1}^{N_c} [||\boldsymbol{\mu}_c - \boldsymbol{x}_i^{(c)}|| - \delta_v]_+^2,$$

$$L_d = \frac{1}{C(C-1)} \sum_{c_A=1}^{C} \sum_{c_B=1, c_A \neq c_B}^{C} [\delta_d - ||\boldsymbol{\mu}_{c_A} - \boldsymbol{\mu}_{c_B}||]_+^2,$$

$$L_r = \frac{1}{C} \sum_{c=1}^{C} ||\boldsymbol{\mu}_c||,$$

where $C$ is the number of fibers in the image and $N_c$ is the number of voxels belonging to fiber $c$. $\boldsymbol{\mu}_c$ is the mean of the embeddings corresponding to fiber $c$, and $\boldsymbol{x}_i^{(c)}$ is the embedding for a single voxel. The parameters $\delta_d$ and $\delta_v$ affect the distances between different clusters and the distances between embeddings belonging to the same clusters, and can be assigned any values deemed appropriate. The total loss for the image is given by the weighted sum

$$L_{img} = \alpha L_v + \beta L_d + \gamma L_r,$$

where $\alpha$, $\beta$ and $\gamma$ can be chosen arbitrarily. Konopczyński et al. used the values $\alpha = 1$, $\beta = 1$ and $\gamma = 0.001$. Denoting the loss for image $i$ in a minibatch by $L_i$, the total loss is obtained as the mean;

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i,$$

where $N$ is the minibatch size. Konopczyński et al. applied the described method to overlapping 32x32x32 patches of their images, and then used a merging algorithm described in the paper to obtain the final segmentation.

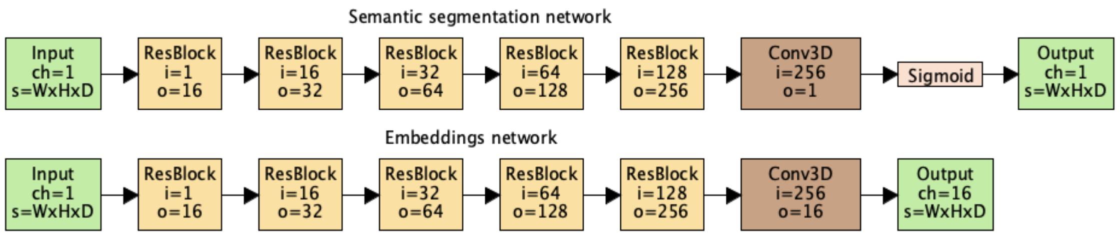


Figure 4: The two network architectures used in ABC, with embeddings of length 16. For each residual block and convolutional layer, the number of channels in the input and output is denoted $i$ and $o$, respectively. The number of channels in the network input and network output is denoted $ch$, and the image size is denoted $s$. A residual block is illustrated in figure 5.

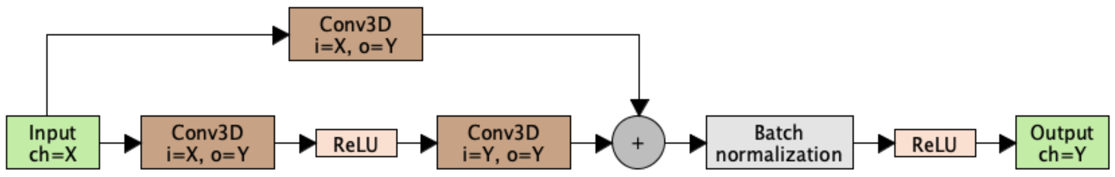


Figure 5: Illustration of a residual block.

# 4 Methodology

## 4.1 Data

Due to difficulties in labeling data, as explained in the following section, a central issue in this project has been the aquisition of training data of adequate quality. Five different data sets have been used to train models, and two additional data sets have been used for validation. The different data sets contain different types of data. First, tomograms of paperboard samples obtained from X-ray microtomography. Second, synthetic data which has been generated algorithmically to resemble real tomograms of fiber networks. Third, a type of 'semi-synthetic' data, and fourth, preprocessed tomography data. The data sets presented below will be named according to what type of data they contain. The data sets containing tomography data will be referred to as data sets **T1** and **T2**, the data sets containing synthetic data will be referred to as data sets **S1** and **S2**. The data sets with semi-synthetic data will be referred to as data sets **SS1** and **SS2**, and the data set containing preprocessed tomography data will be referred to as data set **B**. Each example in the data sets consists of an image, and one binary image for each fiber in the image. All input images are grayscale.

### 4.1.1 Data From Tomography Experiments

Figure 6a shows a 2D slice of a tomogram with voxel size 1.61 $\mu$m and figure 6b shows a slice from a tomogram with voxel size 3.5 $\mu$m. The former comes from the

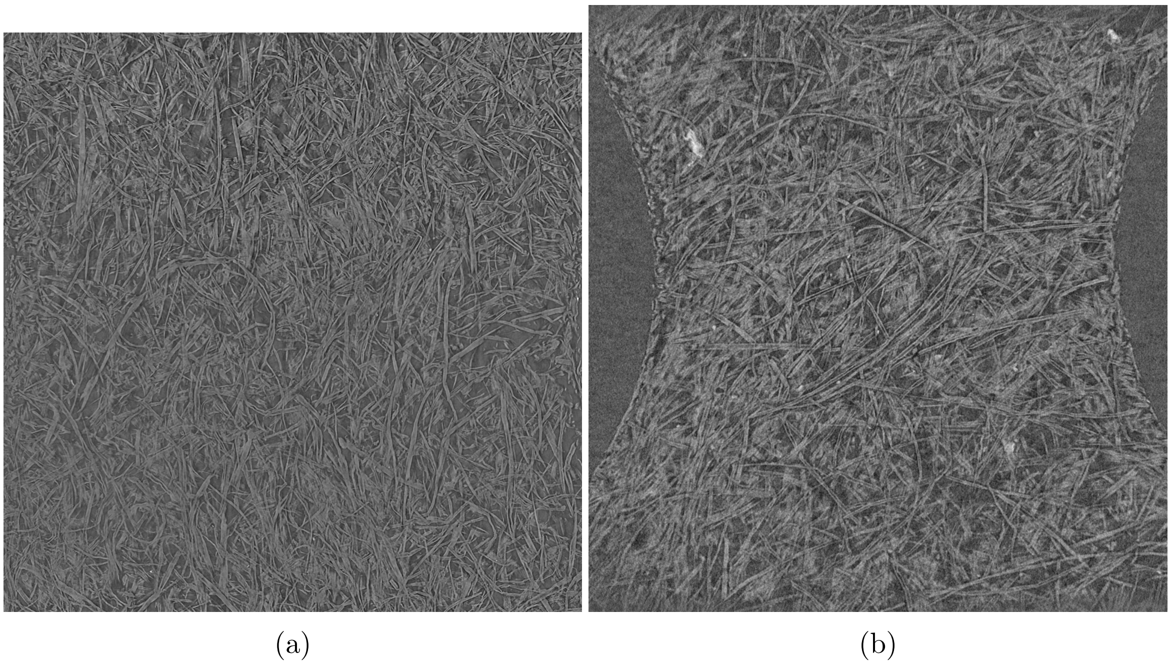


Figure 6: Slices of 3D tomograms with voxel size (a) 1.61 $\mu$m and (b) 3.5 $\mu$m. The images come from the work in [15] and [16], and have been taken using a synchrotron and laboratory-based X-ray tomography, respectively.

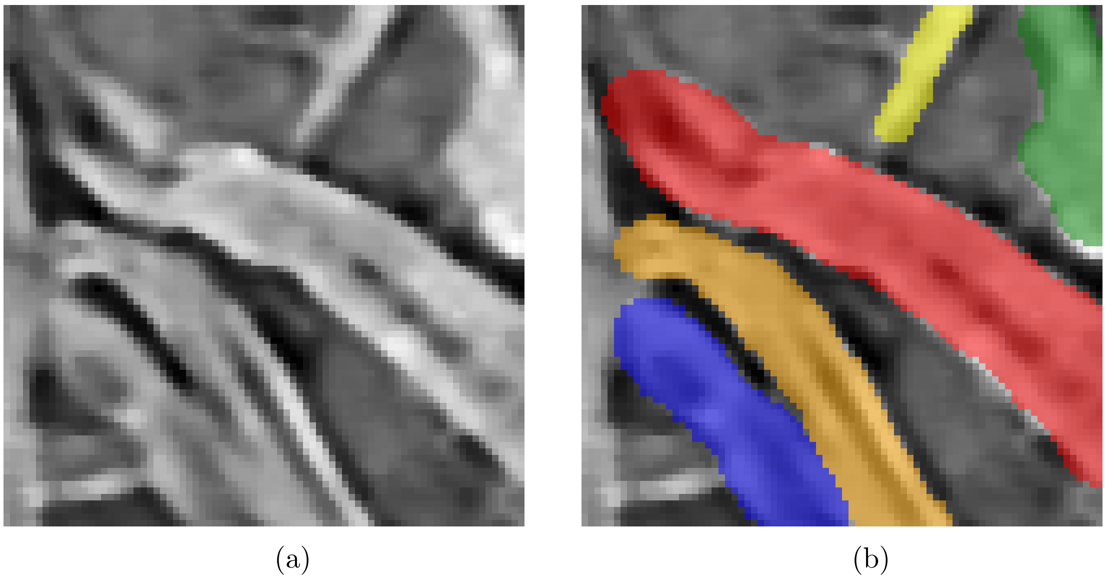


Figure 7: (a) Slice of an image in data set **T1**, and (b) the corresponding labeling. Each color represents one fiber.

experimental work in [15], and the image was taken using a synchrotron. The latter is from the work described in [16], and the image was taken using laboratory-based X-ray tomography. A number of such images from these experiments have been available and used for this project. The tomograms have dimensions 151x1010x964 and

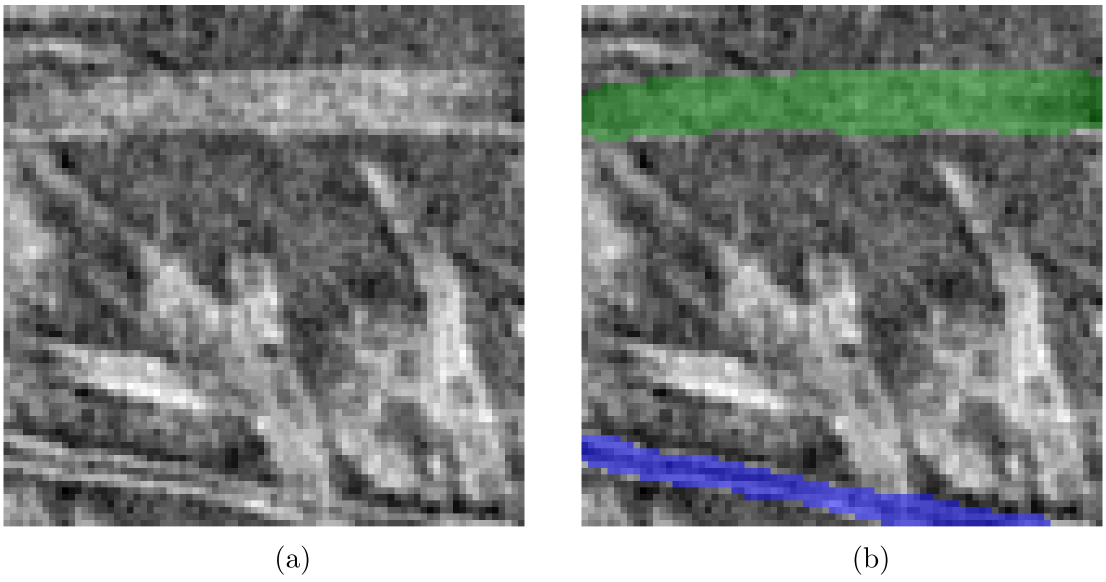


Figure 8: (a) Slice of an image in data set **T2**, and (b) the corresponding labeling.

400x2016x2016, respectively. Due to computational- and memory constraints, smaller sections of such images have been used. Data set **T1** consists of 12 images of size 5x80x80, obtained by selecting patches from tomograms of voxel size 1.61 $\mu$m. Data set **T2** consists of 12 patches of size 5x80x80 from tomograms of voxel size 3.5 $\mu$m. An example of a 2D slice from each data set is show in figure 7a and figure 8a, respectively. Data of higher resolution was available, but such data was presumed to be of too high resolution, and therefor was not used. A 5x80x80 patch would likely be a too small part of the image, not containing enough information.

Data sets **T1** and **T2** were manually annotated using the program ITK-SNAP. Figures 7a and 8a illustrate why the labeling is inherently difficult, even though the images in the data sets were included due to being relatively easy to annotate. In many cases it is difficult to follow a fiber, and see which segments belong to the same one. This means that when manually annotating, care needs to be taken when it comes to selecting which parts of images to label. Data sets **T1** and **T2** have been labeled using slightly different approaches. In **T1**, a higher number of fibers have been labeled, including fibers that are not easily distinguished. In **T2**, the annotation has been more selective, only including more well-defined fibers. Examples of the labeling are shown in figure 7b and figure 8b.

### 4.1.2 Synthetic Data

As described above, a central issue in this project has been the challenges involved in manually labeling data. An alternative approach is to train models using synthetic data, i.e. computer generated images. The idea is that if the generated images are similar enough to the real-world tomograms, the models should be able to perform well when tested on real images. Using synthetic data, it is easy to acquire an arbitrary amount of data, with completely accurate annotations.

Data set **S1** consists of 400 generated images of size 5x80x80. Figure 9 shows an

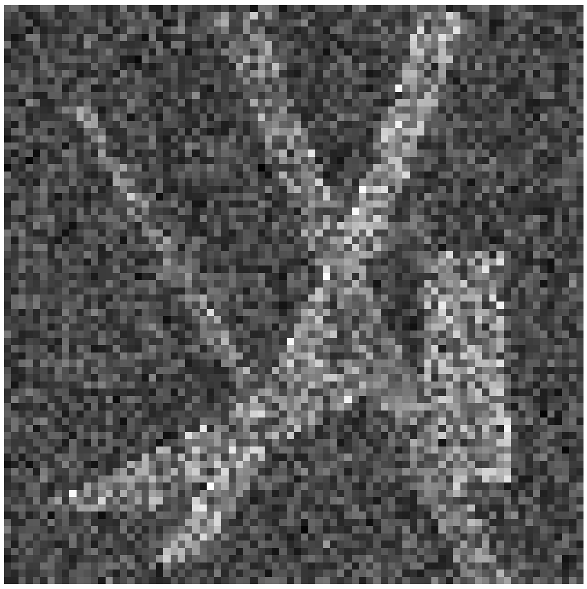


Figure 9: Slice of a synthetically generated image in data set **S1**.

example of a slice of such an image. The images are generated by stacking 2D images on top of each other; First, a 2D image is generated. Each fiber is a circular arc, generated as follows: Coordinates for a circle of a randomly selected radius, centered at the origin, are obtained using the mid-point circle drawing algorithm[17]. Then, a section of this circle is randomly selected and inserted at a random location in the image. The circular arc is repeated horizontally to increase the thickness of the fiber. When a number of fibers have been added, the image is repeated a few times in the depth-dimension, adding depth to the fibers. Two such 3D-images are generated, one of depth 3 and the other of depth 2, and these are stacked two yield a resulting image of depth 5. Finally, random noise is added to the image. This method of generating images implies that all fibers are oriented perpendicularly to the 'depth-dimension'. Data set **S2** consists of 16 images of the same type, and has been used for validation.

#### 4.1.3 Semi-Synthetic Data

In addition to the data types discussed above, we also used a type of 'semi-synthetic' data. This data was constructed using parts of real images. Segments corresponding to single fibers were extracted from tomograms, and these segments were then inserted at random locations in an image with a noisy background. The reasoning behind this approach was that, just like using synthetic data, it could be used to quickly generate a lot of accurately labeled data, but with fibers looking more like those in the real data. Data set **SS1** contains 100 semi-synthetic images of size 5x80x80. Figure 10 shows an example of a 2D slice. Data set **SS2** contains 16 such images, and has been used for validation.

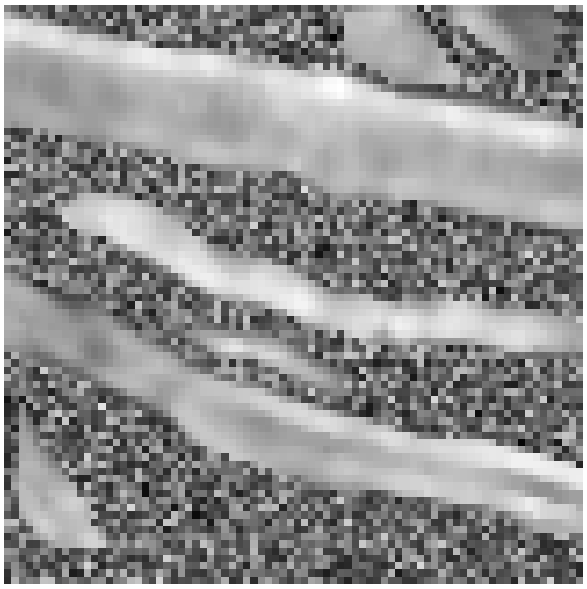


Figure 10: Slice of an image in data set **SS1**.

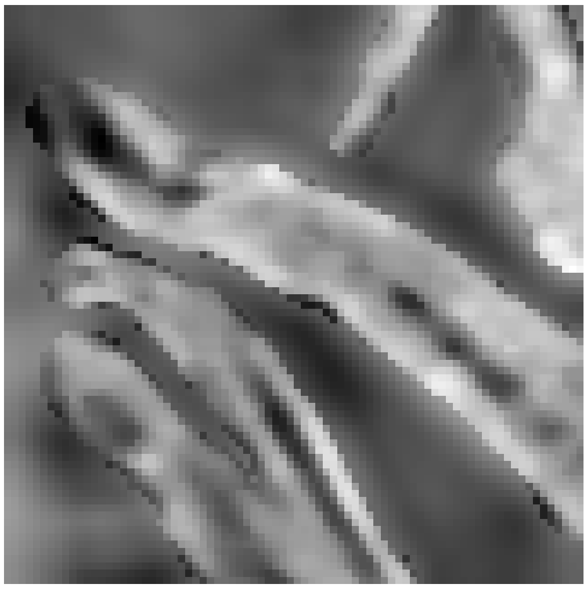


Figure 11: Slice of an image in data set **B**.

#### 4.1.4 Blurred Tomography Data

Data set **B** consists of 12 preprocessed tomography images. This data set is identical to data set **T1**, with the exception that the parts of the images that have not been labeled as fibers have been blurred. The reasoning behind this approach is that some parts of the images which perhaps should have been labeled as fibers but were not, will be blurred out and thereby not confuse the model. An example slice is shown in figure 11. This is the blurred version of the image shown in figure 7a.

### 4.1.5 Data Augmentation

To effectively increase the amount of available data, data augmentation has been applied during training. This means that images have been transformed so that they to some extent appear to be 'new' images. The annotations are transformed accordingly. The following transforms, found in the Volumentations 3D library[18] and referred to by the corresponding class names, have been applied randomly: `Rotate`, `Flip`, `RandomBrightnessContrast`, `GaussianNoise`, `ColorJitter` and `RandomRotate90`. The library has not been developed with instance segmentation in mind, but it is possible to utilize it for this purpose by, instead of passing a semantic mask in the form of a binary image, passing a mask where each fiber is represented by a unique number.

## 4.2 Model Implementations

ABC, as described in section 3.1.2, was implemented using PyTorch. Also, a variant of the method has been implemented and tested. In this variant, the output of the semantic segmentation network has been used as input to the embeddings network. I.e., the embeddings network has essentially been used for instance segmentation of binary images. In this case, the images in the training data have been transformed to binary images, where each fiber voxel has the value 1, and all other voxels has the value 0.

Due to time constraints, as well as poor results in the segmentation of 5x80x80 images, no merging algorithm was tested.

## 4.3 Evaluation

Due to the difficulties in labeling data, and the difficulty of achieving high accuracy, the performances of the models have mainly been evaluated by visualizing the obtained segmentations, including both semantic segmentations and instance segmentations. The uncertainty in how images should be annotated means that evaluation based on a comparison to manual annotations would be misleading. I.e. due to the nature of the problem, evaluation using visualizations is more illustrative than evaluation using metrics.

### 4.3.1 Validation

When training on synthetic or semi-synthetic data, validation data of the same type has been used to obtain a validation loss, i.e. the value of the loss function applied to the validation data. By validating on the same type of data, it is possible to get an idea of how the generalization changes during training, and thereby an indication of what is a suitable number of iterations, even though the performance on the validation data is not representative of how the model would perform on real-world data.

## 4.4 Model Training

The main focus in this work has been the ABC and the two neural networks have been trained many times, using different data sets and parameter configurations. The parameter configurations presented here are the ones that were found to be the most suitable. All models were trained using the Adam optimizer[19]. For all models presented here, the learning rate had an initial value of 0.01, and was decreased by a factor of 2 every 2000th iteration. The models were trained for 10000 iterations. Two NVIDIA A100 GPUs, each with 40 GB memory, were used, using data parallelism. In all cases, the batch size was 16. All input images were normalized to unit variance and zero mean.

The semantic segmentation models (**SM**) will be referred to as models **SM1**-**SM5**, and the embeddings models (**EM**) as models **EM1**-**EM6**. The training- and validation data sets for each semantic segmentation model are presented in table 1. Models **SM1** and **SM2** were trained on real-world tomography data, using no validation data. Models **SM3** and **SM4** were trained on synthetic and semi-synthetic data, respectively. Here, validation data was used to give an indication of how the generalization changes during training. Model **SM5** was trained on data set **B**, i.e. partially blurred real-world data.

| Model | Train data | Val. data |
|---|---|---|
| **SM1** | **T1** | None |
| **SM2** | **T2** | None |
| **SM3** | **S1** | **S2** |
| **SM4** | **SS1** | **SS2** |
| **SM5** | **B** | None |

Table 1: The train- and validation data used for training the five semantic segmentation models. Validation data was used only when training on synthetic or semi-synthetic data. The data sets are described in section 4.1.

| Model | Train data | Val. data | Init. backbone weights | Input type |
|---|---|---|---|---|
| **EM1** | **T1** | None | **SM1** | grayscale |
| **EM2** | **T1** | None | None | grayscale |
| **EM3** | **T1** | None | None | binary |
| **EM4** | **S1** | **S2** | None | grayscale |
| **EM5** | **S1** | **S2** | None | binary |
| **EM6** | **B** | None | None | grayscale |

Table 2: Information about models **EM1**-**EM6**. The backbone of model **EM1** was initialized using segmentation model **SM1**. All other backbones were initialized randomly. The models took one of two types of images as input; grayscale images or binary images.

| $\alpha$ | $\beta$ | $\gamma$ | $\delta_v$ | $\delta_d$ |
|---|---|---|---|---|
| 1 | 1 | 0.001 | 0.01 | 1 |

Table 3: Parameter values used when training the embeddings models **EM1**-**EM6**. These are the values that were found to work best.

Table 2 contains the corresponding information about the different embeddings models. Due to poor performance from the semantic segmentation models trained on data sets **T2**, **SS1** and **B**, these data sets were not used for the embeddings network. The network has been trained on data sets **T1** and **S1**, using both grayscale- and binary images as input, as well as on data set **B** using the grayscale images as input. Note that using data set **B** with binary input would be equivalent to using data set **T1** with binary input, as data set **B** is a partially blurred version of data set **T1**. Initializing the backbone weights using a semantic segmentation network backbone was not found to make a significant, systematic difference. Model **EM1** have been included here to demonstrate this, and the rest of the models have been initialized randomly. All the presented models were trained using the same parameter configuration for the loss function. The parameter values are given in table 3.

## 5 Results and Discussion

### 5.1 Semantic Segmentation

Figures 12b-12f each show a slice of a segmented image of dimensions 5x80x80 and voxel size 1.61 $\mu$m, using the models **SM1**-**SM5**. The original 2D slice is shown in figure 12a. The parts labeled as fibers are marked yellow.

While none of the results are quite satisfying, a few conclusions can be made. It is clear that the type and quality of the training data plays a large role. Model **SM4**, trained on semi-synthetic data, can be seen to perform very poorly, as it labels almost all voxels as fiber voxels. It can be concluded that the type of semi-synthetic data used here is not useful, presumably due to the input images being too dissimilar. It is likely that it is the noise in the semi-synthetic data that is not similar enough to anything in the real data, and that the model has learned to label everything which does not look like such noise as a fiber.

Model **SM5**, trained on partially blurred tomography data, performs better than model **SM4**. The partially blurred tomography data is similar to the semi-synthetic data, in that the images contain 'real' fibers, surrounded by some form of noise. Presumably, the main difference is that the blurred parts are more similar to the real data than the pure noise in the semi-synthetic data.

The models that perform the best are models **SM1** and **SM3**, i.e. those trained on data sets **T1** and **S1**, respectively. Model **SM2**, trained on data set **T2**, does not perform as well as model **SM1**, indicating that the images in data set **T2** were too sparsely annotated. Figure 13 shows a slice of the semantic segmentation of a 5x300x300 image using model **SM3**. On such larger images, models trained on synthetic data have been found to, at least seemingly, perform better than models trained on real-world data. Looking at close-up images such as those in figure 12, it is not obvious that this is the case. To some extent, it appears that model **SM3** simply performs a thresholding operation, labeling mostly light voxels as fiber voxels, whereas model **SM1** is better at identifying also the darker parts of a fiber. However, it may not be necessary to identify all voxels of a fiber for the results to be useful. Hence, to conclude, it is not clear whether the most suitable approach for semantic segmentation is using real-world data or using synthetic data.

## 5.2 Instance Segmentation

Figures 14a-14f each show an instance segmentation of a 5x80x80 image, segmented using models **EM1-EM6**. Each figure shows all five slices of the image. The different colors represent different fibers. The color assigned to each fiber depends on the order in which clusters have been identified by DBSCAN, and has no further interpretation.

Models **EM1-EM5** were used together with semantic segmentation model **SM1**, since **SM1** was the semantic segmentation model that worked best together with these embeddings models. Model **EM6** was used together with model **SM5**, since this gave a slightly better result than using **EM6** together with **SM1**. The $\epsilon$ values used in the DBSCAN clustering are given in the figure captions. In all cases, the *minSamples* parameter was set to 100.

Whereas models trained on data set **T1** and data set **S1** performs similarly well in the case of semantic segmentation, it is clear that the real-world data is more suitable for training the embeddings network than the synthetic data is. This can be seen by comparing the results from models **EM1** and **EM2**, which were trained on real-world data, with those from model **EM4**, which was trained on synthetic data. Also, it can be concluded that using the original grayscale images as input to the embeddings network is better than using binary images; Models **EM3** and **EM5** take binary images as input, and they perform worse than the corresponding grayscale input models.

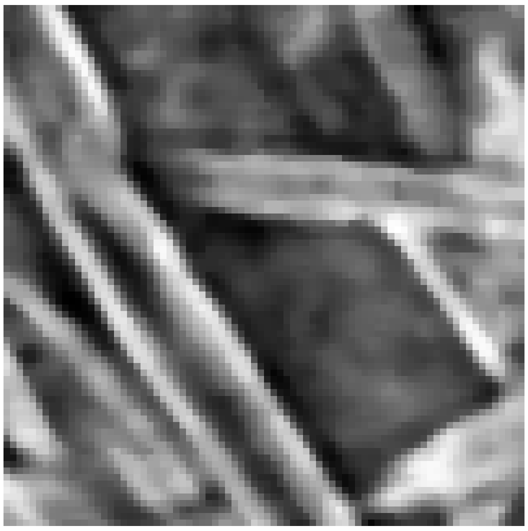
For the example image in figure 14, model **EM1** performs best. However, looking at segmentations of other images it was concluded that there is no systematic difference in performance between model **EM1** and model **EM2**. I.e, initializing the backbone weights using a semantic segmentation model does not make a notable difference. However, the effect such an initialization has may depend on the quality of the semantic segmentation model.

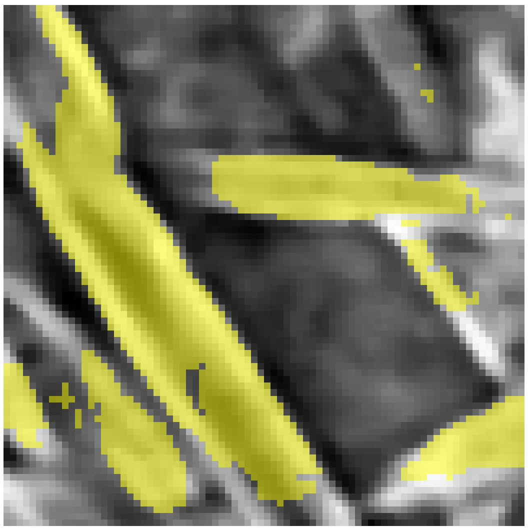
Although models **EM1** and **EM2** give the best results, their performances are not good enough. Compared to when used on other images, the example image in figure

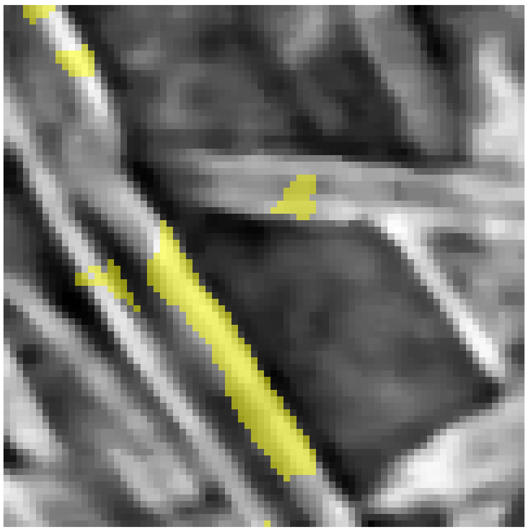
14 is one that the models are able to segment relatively well. See appendix A for more examples of segmentations using model **EM1**. The main reason for the inadequate results is, presumably, the quality of the training data. Better results could possibly be achieved by more extensively labeling real-world data. However, it is highly uncertain if a larger quantity of manually labeled data would make a big difference. Due to the heavy use of data augmentation, the amount of data used here has, effectively, been quite large. An alternative approach to improve the performance of the models is to build on the idea of using synthetic data, by using more sophisticated methods or algorithms to generate data similar to the real-world images.

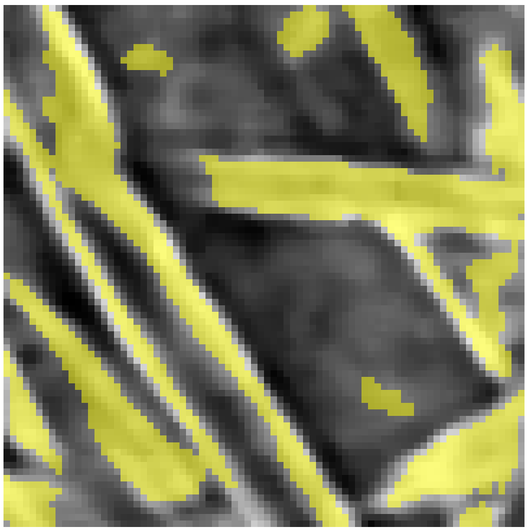
# 6 Conclusions and Future Work

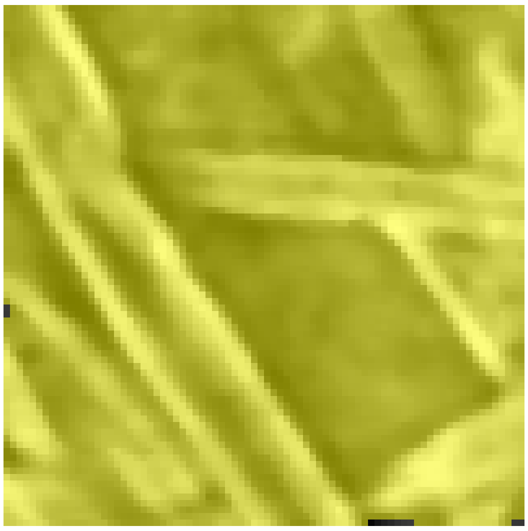
It is not clear that any existing method has the potential to segment tomograms of paper fiber networks with sufficient accuracy. The primary issue is the difficulty in acquiring training data of high enough quality. Two possible ways of approaching this issue is to more extensively manually label data, and developing methods to generate synthetic data more similar to real-world data. Another approach would be to use images of less complex fiber networks, e.g. from a different type of paper, as training data. It is possible that models trained on such data could perform reasonably well also on images of more complex networks. However, it is not clear to what extent it is possible to improve the results using ABC, even with access to large amounts of high quality data. The problem is inherently challenging, and it is possible that more sophisticated segmentation methods are needed.


(a) 2D slice of original image.


(b) **SM1**


(c) **SM2**


(d) **SM3**


(e) **SM4**

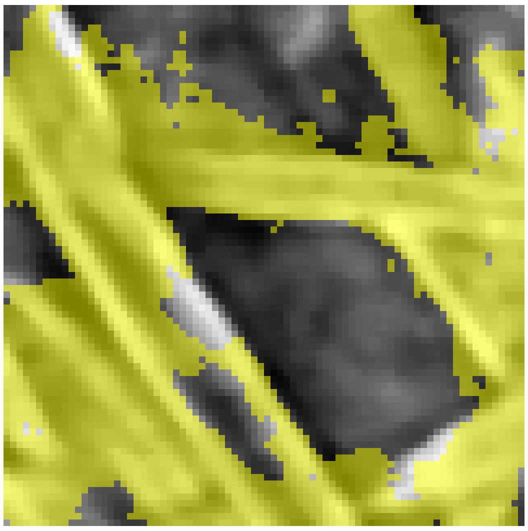

(f) **SM5**

Figure 12: 2D slice of a segmented tomogram, segmented using models **SM1-SM5**.

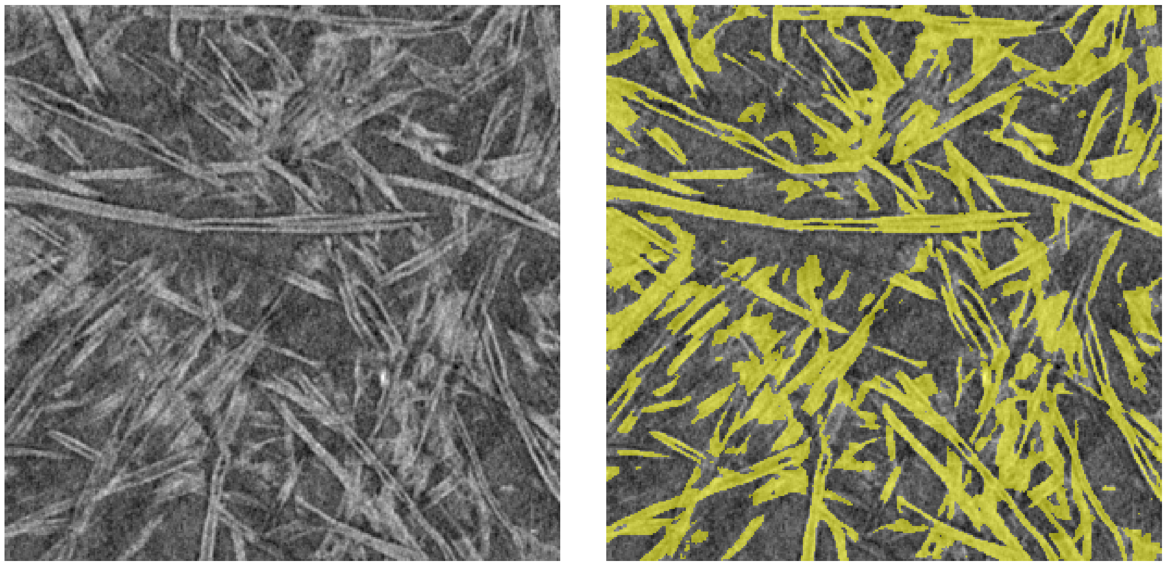
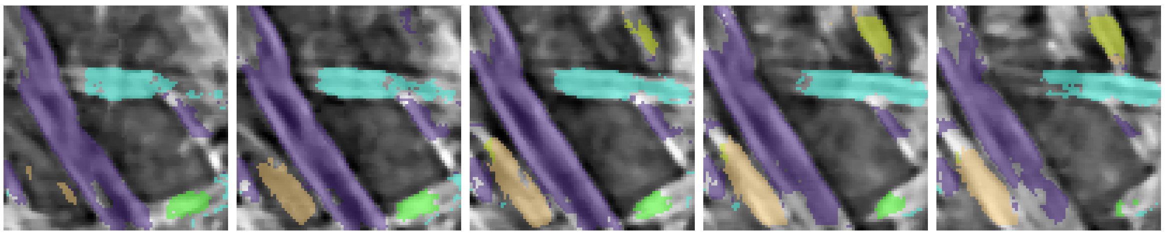
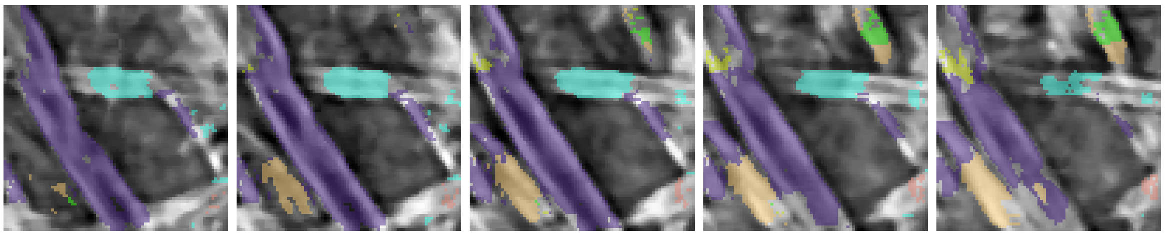


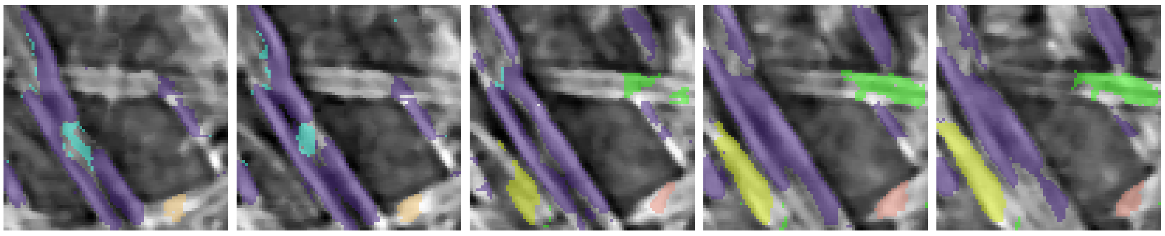
Figure 13: Slice of a semantic segmentation (right) of a 5x300x300 image. The original 2D slice is shown on the left.



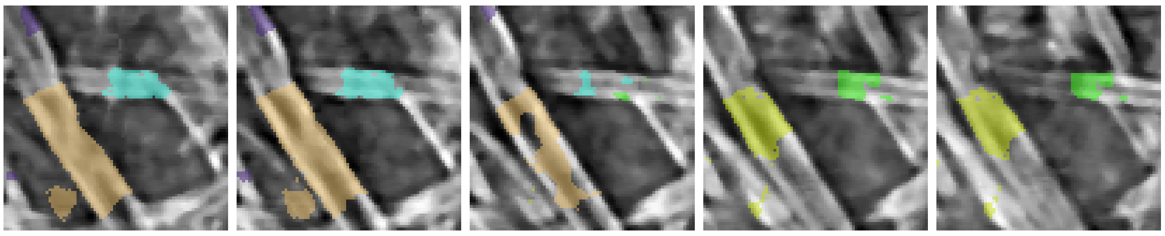
(a) **SM1** and **EM1**, $\epsilon$=0.1



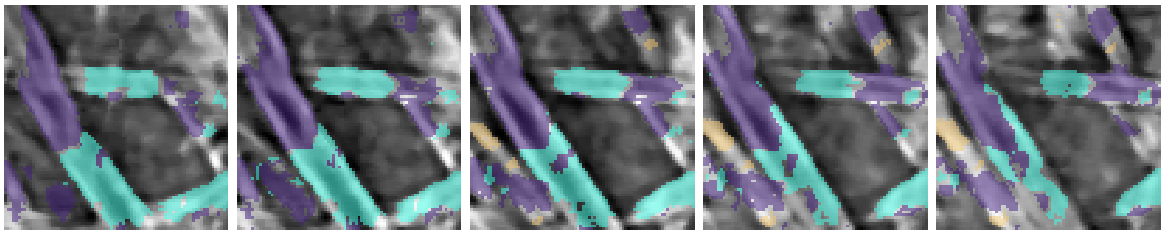
(b) **SM1** and **EM2**, $\epsilon$=0.09



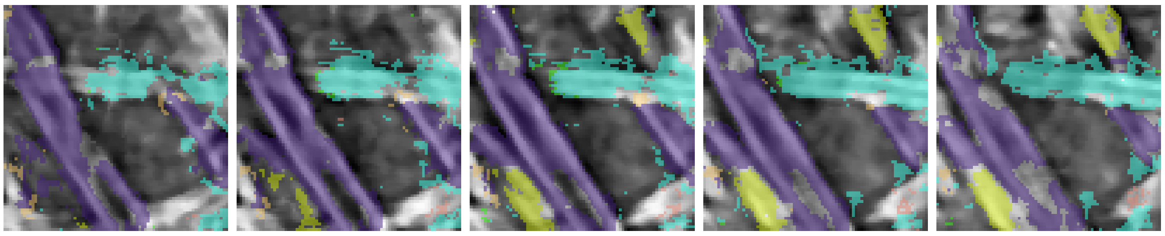
(c) **SM1** and **EM3**, $\epsilon$=0.09



(d) **SM1** and **EM4**, $\epsilon$=0.12



(e) **SM1** and **EM5**, $\epsilon$=0.08



(f) **SM5** and **EM6**, $\epsilon$=0.08

Figure 14: All five 80x80 slices of instance segmentations of a 5x80x80 tomogram with voxel size 1.61 $\mu$m, using embeddings models **EM1-EM6**. Models **EM1-EM5** were used together with segmentation model **SM1**. Model **EM6** was used together with model **SM5**. The $\epsilon$ values are given above.

## References

[1] Mattias Aronsson. On 3 d fibre measurements of digitized paper from microscopy to fibre network. 2002.

[2] Svetlana Borodulina, Erik Wernersson, Artem Kulachenko, and Cris Luengo Hendriks. Extracting fiber and network connectivity data using microtomography images of paper. *Nordic Pulp and Paper Research Journal*, 31:469–478, 09 2016.

[3] Mikko Alava and Kaarlo Niskanen. The physics of paper. rep prog phys. *Reports on Progress in Physics*, 69:669–723, 03 2006.

[4] Aly Badran, Dula Parkinson, Daniela Ushizima, David Marshall, and Emmanuel Maillet. Validation of deep learning segmentation of ct images of fiber-reinforced composites. *Journal of Composites Science*, 6:60, 02 2022.

[5] Yog Sharma, A. Phillion, and D. Martinez. Automated segmentation of wood fibres in micro-ct images of paper. *Journal of microscopy*, 260, 08 2015.

[6] Aly Badran, David Marshall, Zacharie Legault, Ruslana Makovetsky, Benjamin Provencher, Nicolas Piché, and Mike Marsh. Automated segmentation of computed tomography images of fiber-reinforced composites by deep learning. *Journal of Materials Science*, 55:1–17, 12 2020.

[7] Monica Emerson, Kristine Munk Jespersen, Anders Dahl, Knut Conradsen, and Lars Mikkelsen. Individual fibre segmentation from 3d x-ray computed tomography for characterising the fibre orientation in unidirectional composite materials. *Composites Part A: Applied Science and Manufacturing*, 97, 01 2017.

[8] Damian Podareanu, Valeriu Codreanu, Sandra Aigner, Caspar Leeuwen, and Volker Weinberg. Best practice guide - deep learning, 02 2019.

[9] François Chollet. *Deep Learning with Python: Second Edition*. Manning, 2021.

[10] Tomasz Konopczyński, Thorben Kröger, Lei Zheng, and Jürgen Hesser. Instance segmentation of fibers from low resolution ct scans via 3d deep embedding learning. 2018.

[11] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2017.

[12] M. Frei and F.E. Kruis. FibeR-CNN: Expanding mask r-CNN to improve image-based fiber analysis. *Powder Technology*, 377:974–991, jan 2021.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[15] Sara Johansson, Jonas Engqvist, Johan Tryding, and Stephen Hall. Microscale deformation mechanisms in paperboard during continuous tensile loading and 4d synchrotron x-ray tomography. *Strain*, 58, 04 2022.

[16] Sara Johansson, Jonas Engqvist, Johan Tryding, and Stephen Hall. 3d strain field evolution and failure mechanisms in anisotropic paperboard. *Experimental Mechanics*, 61, 01 2021.

[17] Minghua Cao, Siyuan Liu, and Fanghua Cao. Midpoint distance circle generation algorithm based on midpoint circle algorithm and bresenham circle algorithm. *Journal of Physics: Conference Series*, 1438:012017, 01 2020.

[18] Roman Solovyev, Alexandr A Kalinin, and Tatiana Gabruseva. 3d convolutional neural networks for stalled brain capillary detection. *Computers in Biology and Medicine*, 141:105089, 2022.

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

## A Additional Results

Figures A.1-A.5 shows instance segmentations of five different images, segmented using embeddings model **EM1** together with semantic segmentation model **SM1**.

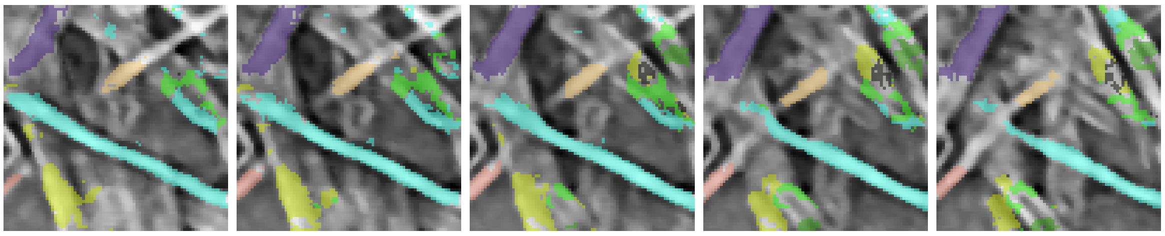


Figure A.1: Instance segmentation of an 5x80x80 image, using model **EM1**, with $\epsilon = 0.09$.

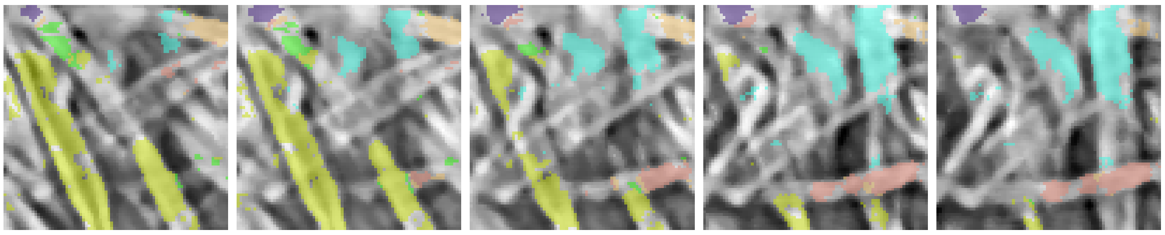


Figure A.2: Instance segmentation of an 5x80x80 image, using model **EM1**, with $\epsilon = 0.1$.

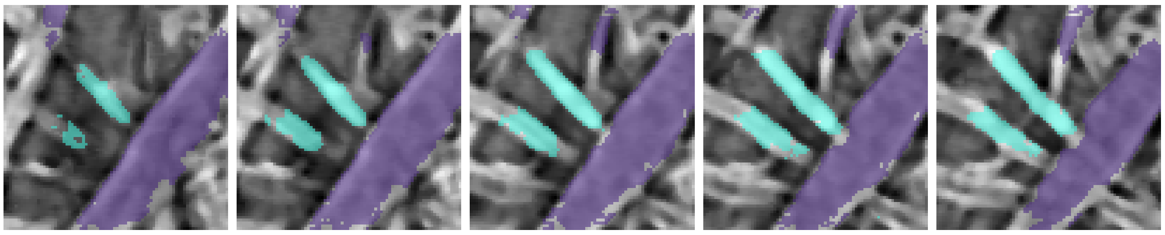


Figure A.3: Instance segmentation of an 5x80x80 image, using model **EM1**, with $\epsilon = 0.07$.

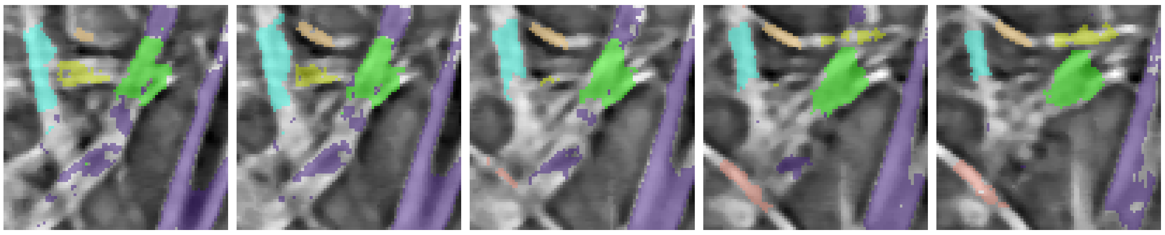


Figure A.4: Instance segmentation of an 5x80x80 image, using model **EM1**, with $\epsilon = 0.09$.

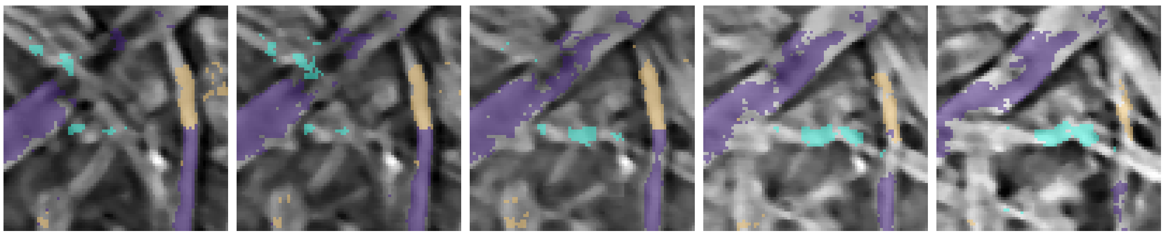


Figure A.5: Instance segmentation of an 5x80x80 image, using model **EM1**, with $\epsilon = 0.09$.