Technische Universität Berlin
Fakultät V - Verkehrs- und Maschinensysteme
Institut für Werkzeugmaschinen und Fabrikbetrieb

Master Thesis

Automated Tracking of Dynamic Microtubules

ZiXuan Liu

Produktionstechnik

Matriculation Number.408014

Berlin 31.05.2022

Supervised by Prof. Dr.-Ing. Olaf Hellwich, Prof. Dr. Guillermo Gallego,
Manuel Thomas Wöllhaf, and Ella de Gaulejac

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

I hereby declare that I wrote this thesis myself using only the appropriately cited literature and auxiliary means.

Berlin, 31.05.2022

Recoverable Signature

X LiuZiXuan

Signed by: Liu ZiXuan

Zusammenfassung

Mikrotubuli sind dynamische Filamente, die in eukaryontischen Zellen vorkommen und bei zahlreichen zellulären Prozessen eine entscheidende Rolle spielen. Um das Verhalten von Mikrotubuli zu untersuchen, ist es heutzutage immer noch üblich, Daten aus Fluoreszenzmikroskopie-Bildern zu extrahieren, indem Kymogramme manuell erstellt und analysiert werden. In dieser Abschlussarbeit haben wir durch die Implementierung mehrerer Bildsegmentierungsmethoden ein Programm entwickelt, das dynamische Mikrotubuli automatisch erkennt und misst. Das Programm ist benutzerfreundlich, erfordert fast keine manuellen Eingriffe und liefert überzeugende Ergebnisse. Der Vergleich verschiedener Segmentierungsmethoden während des Entwicklungsprozesses zeigt, dass die semantische Deep Learning Segmentierung eine bessere Erkennungsfähigkeit und genauere Ergebnisse als Kanten- oder Regionen- orientierte Segmentierungsverfahren der Bildverarbeitung im Falle der Segmentierung kleiner linienförmiger Objekte aus Bildern mit enormen Bildstörungen und unregelmäßigen Rauschmustern aufweist.

Abstract

Microtubules are dynamic filaments that are present in the eukaryotic cells that serve a crucial role in numerous cellular processes. To investigate the behaviors of these microtubules, it has been common to extract data from fluorescence microscopy images by manually generating and analyzing kymographs, and this is still the case. This thesis thus represents an attempt to instead implement a multiple image segmentation method using a novel program that automatically detects and measures dynamic microtubules. This program is user-friendly, requires almost no manual intervention, and has produced some promising results. Comparing different segmentation methods during the development process also revealed that deep learning semantic segmentation offers superior detection capabilities and more accurate outputs than edge-oriented or region-oriented segmentation image processing methods in the case of those small line-shape object segmentations arising from images containing a great deal of noise, particularly here there are irregular noise patterns.

Contents

1.Introduction

1.1 Motivation

Within many biological processes, including intracellular transport, cell motility, and the chromosome segregation seen in most eukaryotic cells, the dynamic polar filaments known as microtubules play an essential role [1]. However, the environment within the cells is too intricate for the direct investigation of microtubule activities in most cases, an in vitro replication of pure tubulin and total-internal reflection fluorescence (TIRF) microscopy have thus commonly been employed to explore various microtubule behaviors.

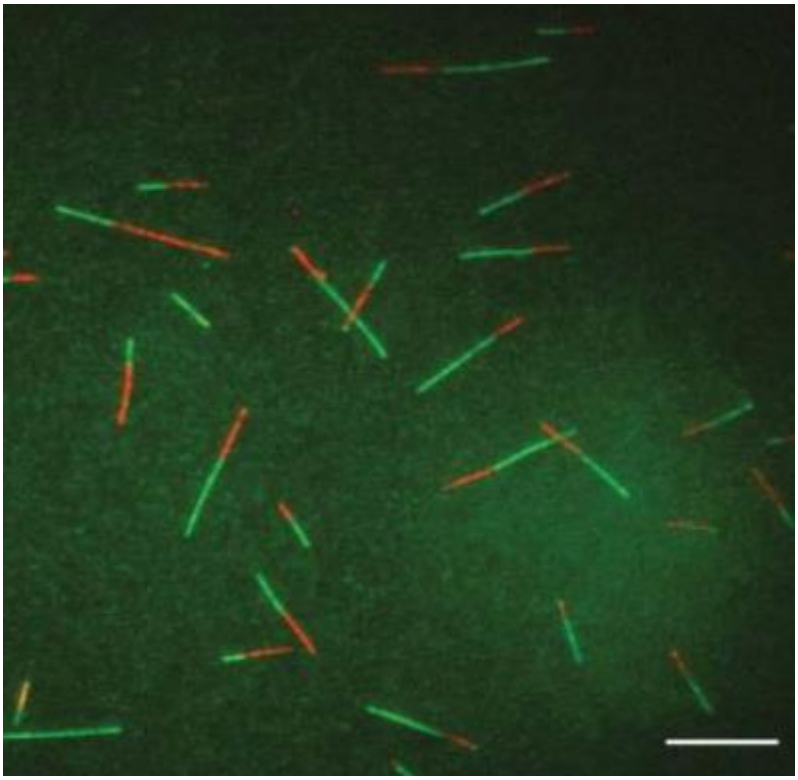

Figure 1. TIRF microscopy image.

Dynamic microtubules (green) grown from stabilized microtubule seeds (red) [1]. Scale bar: 10 µm.

In such in vitro replication, microtubules are laser exposed in a bicolored manner to allow the different components to be distinguished, particularly the microtubule seed (MS) and dynamic microtubule (DM). The MS is shown in Figure1 by the red fluorescence in a portion of each tubule. The MS is a stable microtubule section that is incapable of changing location, shape, or length, while the green fluorescent part of each tubule is the DM, which can expand or contract approximately linearly in conjunction with the orientation of the MS, thereby altering the overall microtubule length.

Deriving data from fluorescent microscopy images by manually creating and analyzing kymographs is still the most common approach to studying these structures [2]. However, in order to analyze and track microtubules both more effectively and automatically, various programs for the automatic recognition and measurement of microtubules, including the collection of statistics about these structures, were developed in this thesis.

1.2 Objective

During in vitro reconstruction, four parameters are essential for describing the behavior of DMs [3]: (1) the polymerization velocity, the rate at which DMs grow (v_g); (2) the depolymerization velocity, the rate at which DMs shrink (v_s); (3) the catastrophe frequency at which DMs switch from growth to shrinkage (f_c); and (4) the rescue frequency, at which DMs switch from shrinkage to growth (f_s) (Figure 2). These four parameters are thus the main objective data that must be obtained to complete observations.

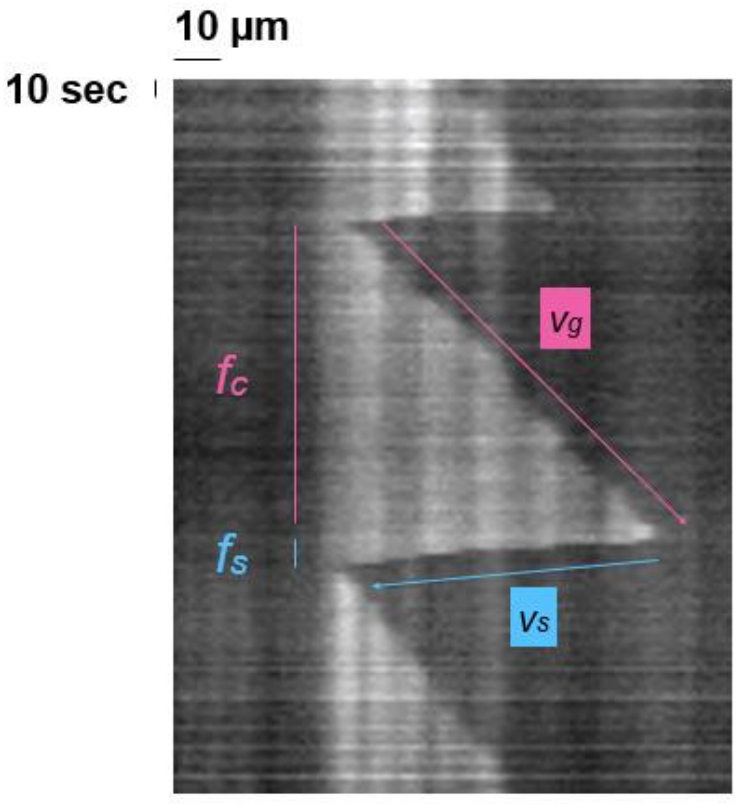

Figure 2. Example of a DM kymograph.

These objective parameters can all be succinctly presented in a kymograph that shows the variation of the length of a DM over time. In Figure 2, the horizontal orientation illustrates a number of frames of the microtubule at 10-second intervals, while the horizontal orientation

shows the length of each frame against a scale bar of 10 μm; the parameters can thus be extracted from the figure as shown, giving v_g, v_s, f_c, and f_s.

1.3 Scope

In this project, various methods of image processing and machine learning approaches were applied to locate and detect the relevant microtubule components, MS and DM; these included edge-oriented segmentation, region-oriented segmentation, and deep learning semantic segmentation. Once the segmentation of MS and DM was achieved, the results were examined for any overlapping issues, which were resolved where necessary by applying overlapping solutions. A tubulin segmentation approach to MS and DM and a concatenation algorithm were then implemented to consistently track and measure the time-variable length data for all DMs. After the length data for the DMs over time were documented, the four parameters v_g, v_s, f_c, and f_s for each DM were generated by means of a piecewise linear regression or Random Sample Consensus (RANSAC) based on the number of local extremes to describe the behaviors of that DM.

1.4 Outline

This thesis is separated into 10 chapters, with the first offering a general introduction to the thesis. The other chapters are then presented as described below:

Chapter 2: This offers an overview of the source data, particularly the raw data used in this work.

Chapter 3: This chapter discusses the fundamentals of the approach and related work. The basic algorithms related to the work are also introduced.

Chapter 4: This chapter explains all of the technical requirements for the project, including both software and hardware.

Chapter 5: This chapter describes the design of the algorithms and segmentation approaches used in the models.

Chapter 6: In this chapter, the implementation process for the program is covered in more detail.

Chapter 7: Evaluation is discussed in this chapter, including any errors, biases, or variance in the results.

Chapter 8: The discussion chapter discusses the results in light of the problems identified during the process.

Chapter 9: This chapter offers a conclusion and summary of the thesis.

Chapter 10: Acknowledgements.

2. Source data

2.1 Raw data clarification and input standardization

For each in vitro reconstitution, the source data consists of two parts: the MS image and a DM video, both supplied in 16-bit 1200 x 1200 resolution using the Nikon NIS-Elements ND2 image format. The MS images and DM videos are taken separately, nevertheless.

In different in vitro reconstitutions, MS and DM may be laser exposed in diverse colors; thus, the use of hue information color-based segmentation methods was not seen as appropriate for this project. To standardize the input images for segmentation processing, as well as incidentally saving on RAM and storage space, all image data in this project were thus converted from RGB mode to grayscale and normalized to 8-bit, creating standardized input data for all the image processing methods implemented in the thesis. This leads to the necessary caveat that all input images in this work are in grayscale.

The MS is the non-dynamic component of a microtubule; thus, one image is sufficient to acquire information on the MS. It is also stationary and can therefore be subject to a long exposure time, which means that MS images are relatively sharp and clear, with comparatively better contrast and tolerable signal-to-noise ratios (SNR), as compared to DM videos, with only a small amount of white “noise” dispersed in the background.

The DM videos were all filmed consecutively after the relevant MS images were taken, allowing information about the length-varying behaviors of the DMs to be recorded. The quantity of frames after decomposition of DM videos was deemed minimal above 40 and maximal below 1,000, at a frame rate of 5 frames per second. The DM video frames all have worse image quality than the related MS images, however, normally featuring lower contrast and SNR, alongside with non-uniform noise; there are also erratic noises in the background of the most of these frame images.

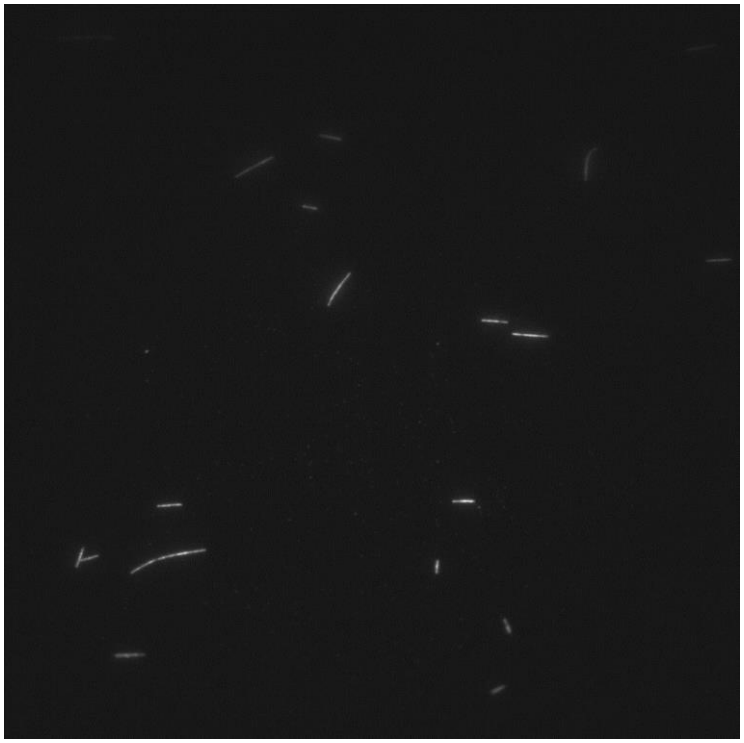

Figure 3. An example of MS image input.

The line-shape white objects are MSs.

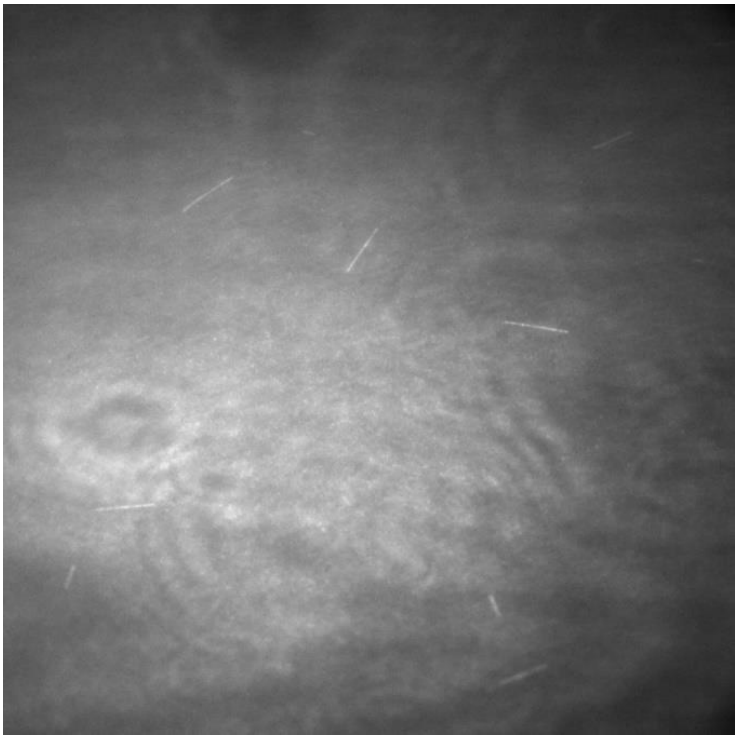

Figure 4. An example of a DM video frame.

The line-shape white objects are DMs.

2.2 Merging MS images and DM video frames

The relationship between MS and DM can be roughly described by considering the endpoint of an MS also being the endpoint of the corresponding DM. To illustrate this relationship, an example of an MS is given in image Figure 3, with a merging substrate; by merging the MS image with the DM frame in Figure 4, graphics can be generated that demonstrate the full-length of the relevant microtubules, as shown in Figure 5(a); Figure 5(b) is the labeled version of (a).

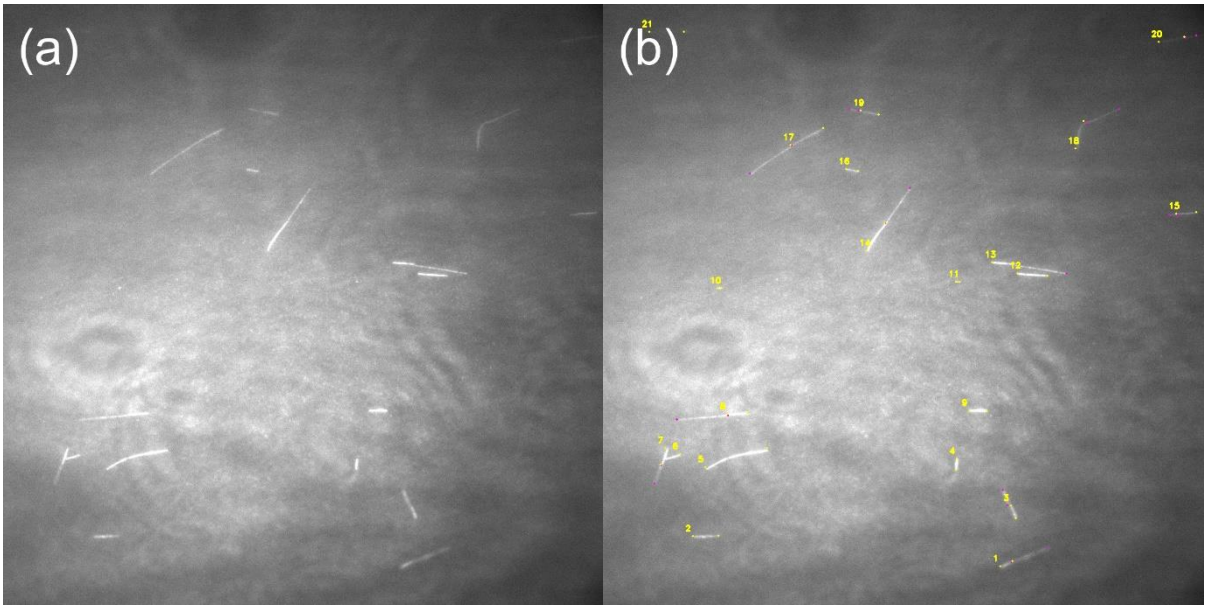

Figure 5. Merging image of MS image and DM frame image.

(a) Merged image from Figure 3 and Figure 4. (b) Labeled version of (a).

In Figure 5(b), the line-shape fragments with yellow dots in the endpoints are MSs, the yellow number tags reference the serial number of those MSs, and the line-shaped fragments with violet dots at the endpoints are DMs. DMs normally appear on one side of the corresponding MSs, with the endpoints from the related MS and DM almost overlapping. Several MSs in the image, including 1, 3, 7, 8, 15, 18, 19, 20, and 21, possess one corresponding DM, while other MSs possess no DMs. In some cases, MSs can have two corresponding DMs, one on each endpoint; in such cases, only data for the longest DM is collected. During the videoing process, the DMs can be seen to change length approximately linearly along the orientation of their corresponding MSs.

Several problems and challenges did emerge during the automatic MS and DM detection and measuring process:

- The shapes of some MSs and some DMs are not perfect line segments; slightly curved segments of tubulin are a possibility.
- During the growth and shrinkage process, some further DMs also become slightly bent.
- A few DMs demonstrate shrinkage behaviors that cause them to vanish, though they may regenerate after a certain period of time.
- Overlapping issues may be encountered in both MS images and DM frame images.

3. Fundamentals and related work

To acquire the required objective parameters, microtubule components must be localized from images and parameterized with respect to their length. The image segmentation methods used should thus be able to provide a solution to localizing the tubules. In visual computing, image segmentation refers to the process of subdividing a digital image into multiple image subregions (collections of pixels) in order to simplify or change the graphical expression of an image to make it more comprehensible and easier to analyze.

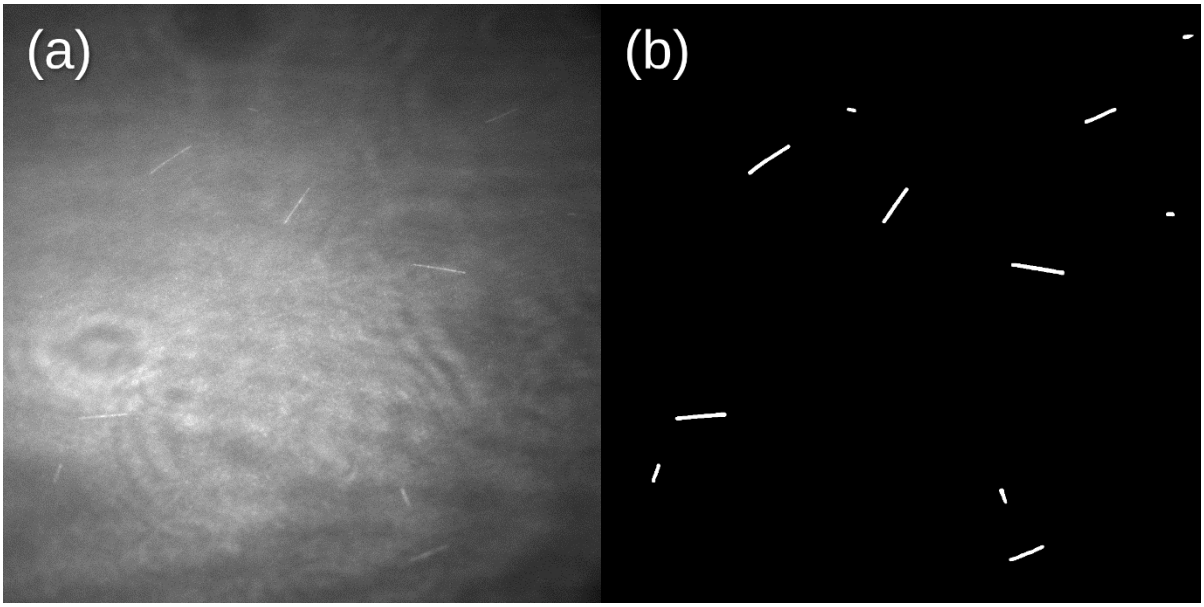

Figure 6. Example of image segmentation.

(a) Example input DM frame image: the line-shaped white objects are DMs; (b) Semantic segmentation deep learning predicted output image from (a); the white line-shaped segments are the predicted segmentations of DMs.

During this thesis programming process, traditional image processing methods such as edge-oriented segmentation (Canny edge detector) and region-oriented segmentation (region growing), as well as a machine learning approach (deep learning semantic segmentation) were applied to segment the MSs and DMs.

3.1 Edge-oriented segmentation

Observing the image data shows that both microtubule components have distinguishable contours; edge-oriented segmentation can thus be used to provide a segmentation solution for MS and DM. However, the relative blurriness and lower SNR in the DM video frames could affect these detection results.

3.1.1 The concept of edges in computer images

Edges are visually distinctive image features that enable the perception of the contours of objects; they are thus among most important features in an image, allowing edge detection algorithms to be commonly used for the segmentation of objects. In terms of image processing, edges are characterized by a radical change in the greyscale or color value of the original image; an edge detector must thus detect variation in grey values and suppress areas with constant grey values.

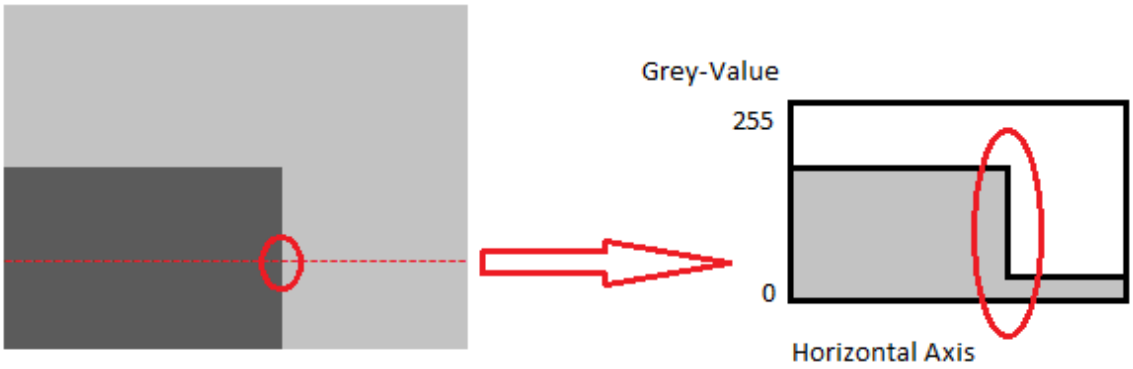

Figure 7. Illustration of edge detector principles.

The grey value along the red dashed line in the left image is recorded to generate the grey value histogram on the right. The edge pixel point inside the left red ellipse represents the drastic change of grey value seen inside the right red ellipse.

From a mathematical perspective, signal changes can be described using the first and second derivatives of the signal: strong changes in the signal lead to large magnitudes of first derivative, while local extreme values in the first derivative cause zero crossings in the second derivative. The first derivative of a signal can be estimated using numerical differentiation, using the general difference quotient with the function $f(x)$, and increment Δx:

$$f'(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

In terms of visual computing, the increment Δx can be considered to represent 1 pixel: inserting this into the general difference equation thus generates the forward difference quotient

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

which corresponds to the convolutional kernel

0	-1	1

However, this forward difference quotient and the corresponding convolutional kernel can only detect signal or grey value variance in one orientation. The heuristic solution is thus to consider the symmetric difference quotient

$$f'(x) = \frac{f(x+\ 1) - \ f(x-1)}{2}$$

which corresponds to the convolutional kernel

1/2

-1	0	1

This convolutional kernel is referred to as a 1-dimensional Prewitt operator, and it can be applied line-by-line to 2-dimensional images to highlight vertical edges in the image. To reduce susceptibility to noise, this operator can also be combined with a smoothing approach process applied to the signal in the edge direction:

1/3

1
1
1

$*$ 1/2

-1	0	1

$=$ 1/6

-1	0	1
-1	0	1
-1	0	1

The scaling factors are usually not noted; however, they should be considered in the final calculation.

3.1.2 Gradients in 2-dimensional graphs

In terms of inspecting directionality when moving from a 1-dimensional greyscale to a 2-dimensional image, the image gradient should be calculated to determine any directional change in intensity or color of images.

In the case of a binary function, the function $f(x,y)$ can be set to have a first-order continuous partial derivative in the plane region D. Then, for each point $P_0(x_0,y_0) \in D$, a gradient vector $f_x(x_0,y_0)\boldsymbol{i} + f_y(x_0,y_0)\boldsymbol{j}$ can be determined; this is referred to as the gradient of the function $f(x,y)$ at the point $P_0(x_0,y_0)$, which can be denoted as $\boldsymbol{grad}\, f(x,y)$ or $\nabla f(x,y)$,

$$\nabla f = \frac{\partial f}{\partial x}\boldsymbol{i} + \frac{\partial f}{\partial y}\boldsymbol{j}$$

In a flat computer image, the plane region, D, can be set as two dimensional along the x and y axes, with a discrete area, and the partial derivatives taken along the x-axis and the y-axis deliver gradient G_x in the x-direction and gradient G_y in the y-direction. By combining both directional gradients, G_x and G_y, the magnitude $|G|$ and direction θ of the gradient G can be calculated:

$$\text{Magnitude: } |G| = \sqrt{{G_x}^2 + {G_y}^2}$$

$$\text{Orientation: } \theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Applying gradient-based edge detection in this way can improve isotropic detection ability.

3.1.3 Canny edge detector

The Canny edge detector [4] is an operator that applies a multi-stage algorithm to detect a wide range of edges in pictures. The process used by the Canny edge detector has four phases:

1. Image Smoothing: In order to reliably detect edges even in noisy images, the image is first smoothed with a Gaussian filter. As part of this process, the size of the convolution mask, as well as the variance parameter, σ, can be adjusted as required, as the more smoothing is done, the fewer edges can be detected.

2. Gradient Determination: This involves the application of operators such as the Sober operator in the x and y direction of the convolution computation, such that the gradient magnitude, $|G| = \sqrt{{G_x}^2 + {G_y}^2}$, and the gradient orientation, $\theta = \tan^{-1}(\frac{G_y}{G_x})$, can be calculated. Rounding of the gradient orientations to 0°, 45°, 90°, and 135°, respectively is also completed at this stage.
3. Suppression of Non-maxima: As the edge point should be where the filter response is strongest, all gradient magnitudes that are not a local maximum are suppressed (set to 0), before the edge is reduced to exactly 1-pixel width.
4. Hysteresis for Binarization of the Image: Upper-and-lower-bound thresholds are used to determine which local maxima are actual edge pixels and which occur only due to noise; any irrelevant edges are then suppressed. The parameters for the upper bound and lower bound are adjustable, to permit customization.

3.2 Region-oriented segmentation

Region-oriented segmentation requires the pixels of the object to be connected in a predefined manner: each segment should thus be a complete region, with the segments between each object being disjoint. In the case of microtubule component detection, almost all MSs are scattered across various locations of the image, with minimal junctions or no junctions whatsoever; the DM frame images produce a similar situation, and thus most cases of MS and DM segmentation fit the required usage scenario for region-oriented segmentation.

The region-oriented segmentation method focuses on identifying similarities between contiguous pixels to obtain a compact representation. Beginning with the selection of a given set of seed points, the algorithm thus groups those adjacent pixels that possess similar attributes into unique region segments. Grey-level intensity is the most prevalent means used to judge this similarity, though other methodologies based on other graphical characteristics such as variance, entropy, and multispectral features may also be valid, depending on the specific image data [5].

The following steps are used to apply a basic computing algorithm for region growing and flood fills:

1. Choose a set of seed points based on the objects and background distributions in the image such that each point is locate inside an object: the region growing process begins from the location of these seed points.
2. Select the growing criterion that will be used to decide how the regions append adjacent pixels during the region growing process. Grey value difference, pixel intensity, texture, and color are common choices for this growing criterion.
3. Determine the growing neighborhood paradigm. Patterns with 8-connected neighborhoods and 4-connected neighborhoods, as illustrated in Figure 8 are commonly used.

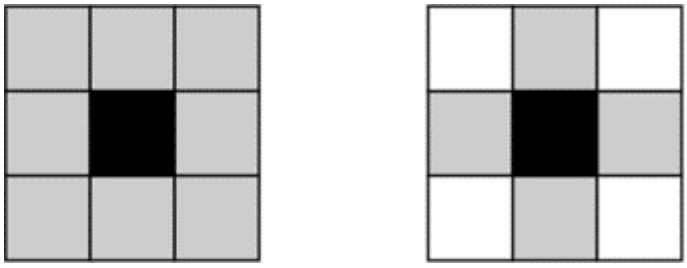

Figure 8. Examples of 8- and the 4-connected neighborhoods.

The black pixel represents the seed pixel, while grey pixels indicate growing neighborhoods that will be associated with it. The left-hand figure shows an 8-connected neighborhood, while the right-hand figure shows a 4-connected neighborhood.

4. Execute the growing process for each seed point, expanding the seed point into a region by associating it with the growing neighborhood pixels that fit the criterion selected in step.2.
5. Take the grown regions in step 4 as new seed regions, and repeat the growing process for each new seed region.
6. Iterate steps 4 and 5 until no growing neighborhoods fit the growing criterion or the region reaches the image boundary.

Algorithm 1 Region growing

Input: image, SeedPoint, thres

```
label ← 1
MarkMap ← binary image/boolean image initialized with all 0 at the size of the input image
MarkList ← empty list
MarkList.append(SeedPoint)
MarkMap[SeedPoint] ← label
while MarkList is not empty do
    CurrentPoint ← MarkList.pop(0)
    Neighbor ← surrounding 4 or 8 pixels of CurrentPoint
    for each Neighbor of CurrentPoint do
        if grayvalue differ < thres and Neighbor in image boundary then
            MarkList.append(Neighbor)
            MarkMap[Neighbor] ← label
        end if
    end for
end while
```

Output: MarkMap

Figure 9. Pseudocode of the region growing algorithm.

Inputs: image is the input image to be segmented; SeedPoint is the starting point for the algorithm; thres is the threshold of the gray value difference.

To implement a region growing algorithm in this case, two options were feasible with respect to selecting the initial point: (1) Once location information about the MSs was extracted from the MS image, the endpoints of all MSs could be used as the seed points for a region segmentation process for the DMs, allowing all DMs to be segmented, or (2) One seed point within the background, for example (0,0), could be selected, allowing the background to be segmented out, producing results equivalent to the segmentation of all DMs.

3.3 Deep learning semantic segmentation

Noise in both MS images and DM video frames has a strong negative impact on the detection and measurement of microtubule components. To segment the MS and DM accurately, deep learning semantic segmentation is thus proposed, utilizing the ability of deep learning to solve complex problems to potentially generate improved.

3.3.1 Semantic segmentation

Semantic segmentation is a fundamental task in visual computing, involving separating visual inputs into different, independently semantically interpretable, categories. This involves taking raw data, such as flat images, as input and dividing these into different highlighted areas of interest, with each area assigned a category ID according to the object it belongs to. In Figure 10 for example, it may be necessary to distinguish all pixels in the image that belong to an automobile and to classify these within the “car” category by applying blue masks while identifying pixels belonging to pedestrians as “pedestrian” using red masks, and so on.

Figure 10. Example semantically segmented image.

Early work on image problems identified only edges, such as lines and curves, or gradient elements, providing pixel-level image understanding rather than reflecting the ways humans perceive images. Semantic segmentation brings together various image parts belonging to the same target to address this problem, thereby expanding its fields of application [6].

3.3.2 Convolutional neural networks for semantic segmentation

For image classification, the fundamental architecture of a convolutional neural network model is constructed of convolutional layers, pooling layers, and a fully connected layer, as demonstrated in Figure 11. The convolutional layer searches the image by applying several adaptive filters for the extraction of image features and thus builds up a feature map using various convolution operators. The pooling layer then applies down-sampling, grouping these features together for simplification, thus compressing the information from the previous layer. The fully connected layer can then apply a classification step based on mapping the spatial tensor from the previous layer to a fixed-length vector. To gain more information from the input image, however, multiple convolutional- and pooling- layers should be added to the network structure (Figure 12), as the initial convolutional- and pooling-layers can then be used to extract features such as edges and colors over a small area, while the additional layers can

obtain higher-level features across a larger area. The feature extraction ability can also be increased by appending further convolutional- and pooling- layers, creating a structure referred to as a deep neural network or deep learning.

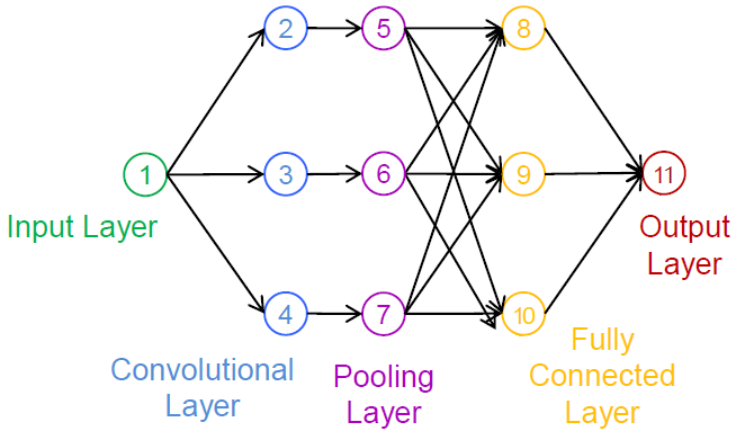

Figure 11. Simple convolutional neural network structure.

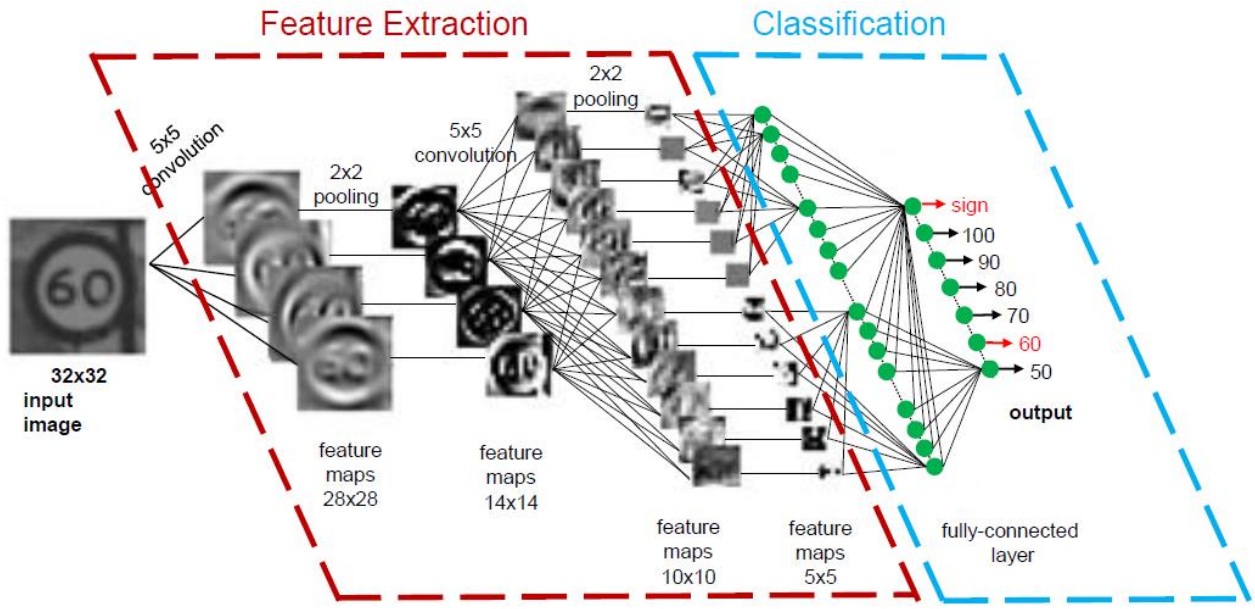

Figure 12. Deep learning neural network.

Here, the architecture combines multiple convolutional layers and pooling layers [7].

In semantic segmentation the feature extraction layer retains the same structure and has the function of down-sampling; however, its output is in the form of segmentation space maps of the same size as the input image, thus retaining spatial information; based on this, instead of using the fully connected layer, the convolutional layers are combined with the pooling layers to work as an up-sampling mechanism to increase the spatial tensor size, thus increasing the resolution and producing a fully segmented image. This form of neural network architecture is

thus known as an encoder-decoder structure (Figure 13). In the decoder, the neural network calculates the possibility of each pixel belonging to a given category; based on correct setup of the probability threshold, it is thus straightforward to distinguish which pixel points can be attributed to which category.

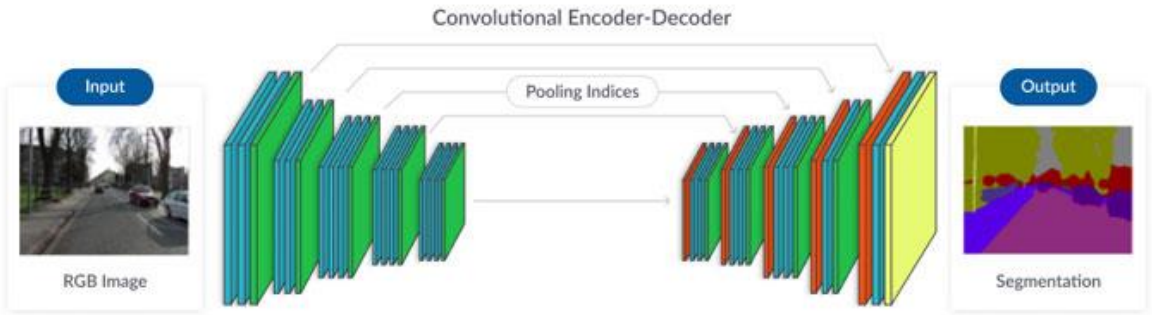

Figure 13. Structure of deep learning semantic segmentation

The left part of the neural network is the encoder, while the right part of the neural network is the decoder [8].

Simple stacking of the encoder and decoder layers could cause information loss during the down-sampling process, however, in order to compensate for this loss, a skip connection is added in the network [9], which allows the decoder to access the features produced by the encoder layers. This may be accomplished by concatenation between intermediate layers of the decoder and intermediate layers of the encoder at appropriate positions.

3.4 OpenCV minimum area rectangle

Once segmentations of the microtubule components are obtained, a substitute model for such segmentation is needed to allow derivation of the lengths and endpoints of the relevant MS and DM portions. For this thesis, the algorithm minimum area rectangle and its corresponding image processing function in OpenCV were applied to modularize the segmentations of microtubule components into line-shaped rectangles in order to calculate the respective lengths and endpoints.

The program function cv2.minAreaRect() is derived from polygon generalization, also known as the smallest surrounding rectangle problem. This is used to find the minimal area rectangle that can box in the segmentations, and three mathematical theorems [10] are thus essential to this algorithm:

Theorem 1: For any rectangle with four points arbitrarily chosen so that its edges do not contain more than one point, there exists another rectangle such that each of its edges

contains one, and only one, point; the area of that rectangle must be smaller than the area of the initial rectangle [10].

Theorem 2: One side of the minimum-area rectangle enclosing a convex polygon is collinear with one of the edges of the polygon [10].

Theorem 3: The minimum-area rectangle encasing a basic, closed, chain-coded curve's convex hull is identical to the minimum-area rectangle encasing that curve [10].

After segmentation, based on the three theorems above, the minimum area rectangle algorithm was thus programmed using the following steps [11]:

1. Identify the convex points of the segmentation to be analyzed by applying an exhaustive search algorithm; the corresponding convex polygon is then generated based on the connections between two adjacent convex vertices.
2. Use the rotating calipers algorithm [12][13] to iterate over the edges of the convex polygon to compute the smallest bounding rectangle with an edge coincident with each polygon edge.

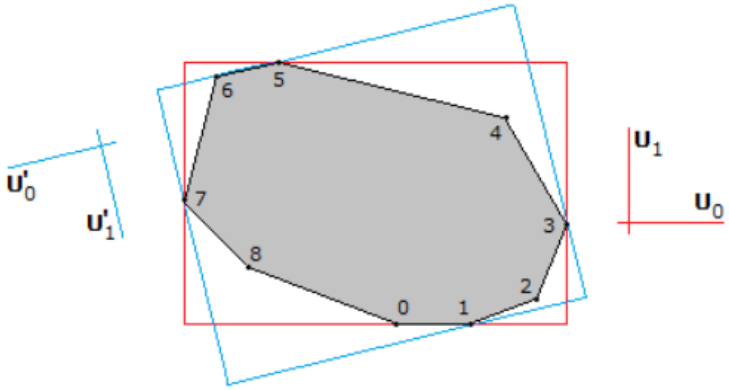

Figure 14. Schematic representation of bounding rectangles.

There are 8 polygon convex vertices in this case. The initial rectangle is in red, with axis orientation U_0 and U_1, supported by edge $< 0,1 >$. The next rotation, the minimum rotation angle as determined by edge $< 5,6 >$, is demonstrated in blue, with axis directions U_0' and U_1' [11].

3. Calculate the dimensions of the computed rectangles: the one with the minimum area is the minimum-area rectangle that encircles the segmentation.

3.5 OpenCV approximate polygonal curve(s)

Image data in this case may suffer from the problem of tubulin overlapping. To resolve this issue, the OpenCV approximate polygonal curve(s) approach was adopted to locate the endpoints of overlapping tubulins.

The function cv2.approxPolyDP generates a curve or a polygon that approximates another polygon with fewer vertices such that the distance between them is smaller than or equal to the specified precision. For this approximation process, the Douglas-Peucker algorithm [14] is employed.

Given a polygon to be approximated and a threshold parameter ε, the specific implementation logic of the algorithm is as follows:

1. Locate the pair of pixel points, A and B, on the contour of the polygon that has the largest Euclidean distance by applying an exhaustive search algorithm.
2. Separate the contour of the polygon by extrapolating these two pixels into two trajectory curves, TJ_1 and TJ_2.
3. Consider the first of the trajectory curves TJ_1, and connect an auxiliary line AB between the two points A and B that form the chord of the two trajectory curves.
4. Iterate through all other points on trajectory curve TJ_1, and thus calculate the distance from each point to the line AB. Denote the point with the maximum distance as the splitting point, C. The maximum distance is then recorded as MD.
5. Compare the distance MD with the pre-defined parameter ε; if $MD < \varepsilon$, then the line AB is an appropriate approximation to the trajectory curve TJ_1.
6. If $MD \geq \varepsilon$, divide the trajectory curve TJ_1 by using the splitting point C to separate it into two segments TJ_{AC} and TJ_{CB}. Repeat steps 3 to 5 for both segments.
7. After step 6, if there exists still a sub distance $MD'(from\, TJ_{AC}\ or\ TJ_{CB}) \geq \varepsilon$, repeat steps 3 to 6 until all sub distances are smaller than ε.
8. The approximation curve of the trajectory curve TJ_1 is then formed by connecting the splitting points in sequence.
9. Repeat the process from step 3 to step 8 for the trajectory curve TJ_2 to obtain its approximation curve.
10. Once the approximation curves for both TJ_1 and TJ_2 are acquired, combine these to achieve the approximation polygon.

3.6 Random sample consensus

In some cases, calculation of objective parameters after obtaining information on the length of microtubule components requires the application of a random sample consensus (RANSAC) algorithm.

RANSAC uses an iterative approach to estimate the parameters of a mathematical model from a set of observed data containing outliers. RANSAC is a non-deterministic algorithm in the sense that it produces a result that is reasonable within a certain probability, while additional iterations increase the magnitude of this probability [15].

The basic premises of dataset suitability for RANSAC are: (1) the existence of inlier data that can be described by several sets of model parameters, allowing outlier data to be considered as noise and excluded from modeling; and (2) the existence of a procedure or a program for estimating the parameters of a model that will best interpret or fits this data.

Once input data matching the premises exists, given a threshold number N that is smaller than the quantity of data, RANSAC can generate results using the following iterative steps:

1. Randomly select a small group of data points and consider this subset of data to be a set of hypothetical inliers.
2. Calculate a model and its parameters that fit the selected hypothetical inliers.
3. Compute the rest of the data points to verify whether they fit the assumed model; those data points that fit the model well are referred to as the consensus set.
4. Record the inlier data point quantity of the consensus set.
5. Repeat the steps above until the estimated model is sufficient that the inlier data quantity reaches the threshold number N.

The determination of algorithm parameters plays an important role in the results of this process. Assuming that the probability ω represents each time a single selected data point belongs to an inlier,

$$\omega = \frac{inlier\ quantity\ in\ data}{total\ quantity\ of\ data\ points}$$

Thus, if in each calculation the quantity of n data points needed to estimate a model is chosen independently, based on the definition of ω, the probability that at least one outlier will fall within the selected n data points is

$$1 - \omega^n$$

In the case of k iterations, the probability that at least one outlier is sampled in each of the k iterations used to compute the model can be represented as

$$(1 - \omega^n)^k$$

Thus, the probability P that an accurate model can be computed by sampling n inlier points is

$$P = 1 - (1 - \omega^n)^k$$

The value of ω is normally considered a prior value; however, it may be unknown initially. If the value of ω is not known in advance, an adaptive iteration method can be used, with an infinite iteration number set at the beginning, and the posterior value of the last iteration or so used to determine current inlier ratio, with this acting as the prior value ω in the next iteration, allowing updates to the model parameters.

All these algorithm processes can be implemented using the Python package sklearn.linear_model.RANSACRegressor [15].

3.7 Piecewise linear regression

In most circumstances, piecewise linear regression is required to calculate the four required objective parameters, based on information about the length of the microtubule components.

Piecewise linear regression, also known as segmented linear regression or broken-stick linear regression, is a regression estimation method that is applied when the regression of the dependent variable, y, on the independent variable, x, obeys a certain linear relationship in the x range and a linear relationship with the different slopes in other ranges. This method thus uses the indicator variable to fit the regression model for each segment (different range) of the data simultaneously. Breakpoints are defined as those values of x where the slope of the linear function changes (shown as a red dot in Figure 15), and the value of a given breakpoint may or may not be known prior to analysis; however, in general, it will be unknown and will have to be estimated. A best-fitting method is performed using the least-squares method as in normal linear regression; however, in this variant of linear regression, the whole dataset is replaced with a single line segment.

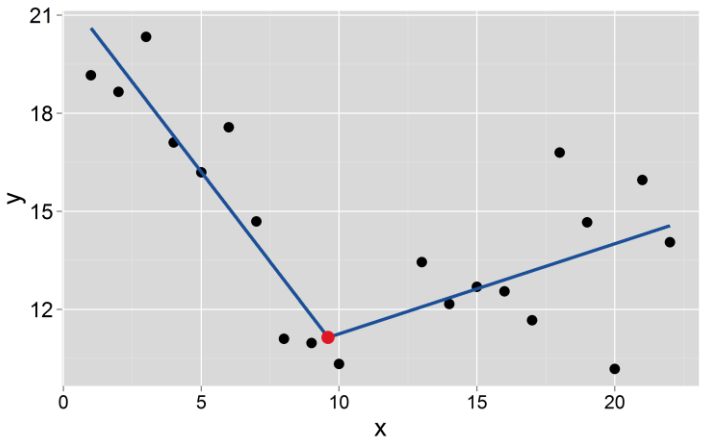

Figure 15. Example of segmented linear regression.

The x indicates the independent variable and the y represents the dependent variable; the black dots are data, while the red dot is the breakpoint of two data segments, and two blue lines are the linear regression in the corresponding segment.

In an example of two segment linear regression where there is only one breakpoint, assuming the breakpoint is at $x = bp$, the two segment model can be written as

$$y = k_1 x + b_1 \qquad for\ x < bp$$

$$y = k_2 x + b_2 \qquad for\ x \geq bp$$

At the breakpoint, the two equations must have the same dependent variable y in order to maintain continuity of the regression function; thus

$$k_1 bp + b_1 = k_2 bp + b_2$$

allowing the piecewise linear regression to be re-written as

$$y = k_1 x + b_1 \qquad for\ x < bp$$

$$y = k_2 x + b_1 + bp(b_1 - b_2) \qquad for\ x \geq bp$$

Further fitting of the model is achieved by applying the least squares method, which minimizes the sum of the squares of the residuals (SSD). This can be done by defining any node i in the observed values as y_i and its fitted value as t_i, as provided by the models. The minimization of SSD is thus given by the equation

$$\min \sum_{i=0}^{n} (y_i - t_t)^2$$

Solving this equation will therefore generate the requested piecewise linear regressors.

The total process of segmented linear regression is integrated into the Python package as piecewise linear functions (pwlf) [16].

4. Technical Requirements

4.1 Hardware requirement

A number of standard image processing methods, such as the Canny edge detector and region growth, were used in this work to achieve the detection and segmentation of MS and DM. These software approaches demand only moderate PC performance, while the training process required for deep learning convolutional neural networks, on the other hand, requires a high level of computational performance. The computer hardware selected thus needed to meet the requirements for neural network training. For the reproduction of the thesis procedure including the training of the neural network as needed, the recommended hardware environment is thus

- Graphic card VRAM >= 8GB (GTX 1070 or better)
- Swap partition >= 32GB
- System ram >= 64GB
- Hard drive space >= 40GB
- CPU clock speed >= 3GHz

4.2 Software environment

The program was written in Python and the software environment set up as noted below:

- GPU driver: nvidia-driver-450 (or newer)
- TensorFlow version: tensorflow-gpu==2.2.0 / tensorflow==2.2.0
- CUDA driver: V10.1
- Anaconda python-3.7

5. Model and algorithms

5.1 Semantic segmentation neural network structure

The structure of the neural network had to be determined prior to the training process. As U-Net has excellent suitability for biomedical application scenarios [17], this was proposed as the basic architecture of the deep learning semantic segmentation convolutional neural network, as illustrated in Figure 16.

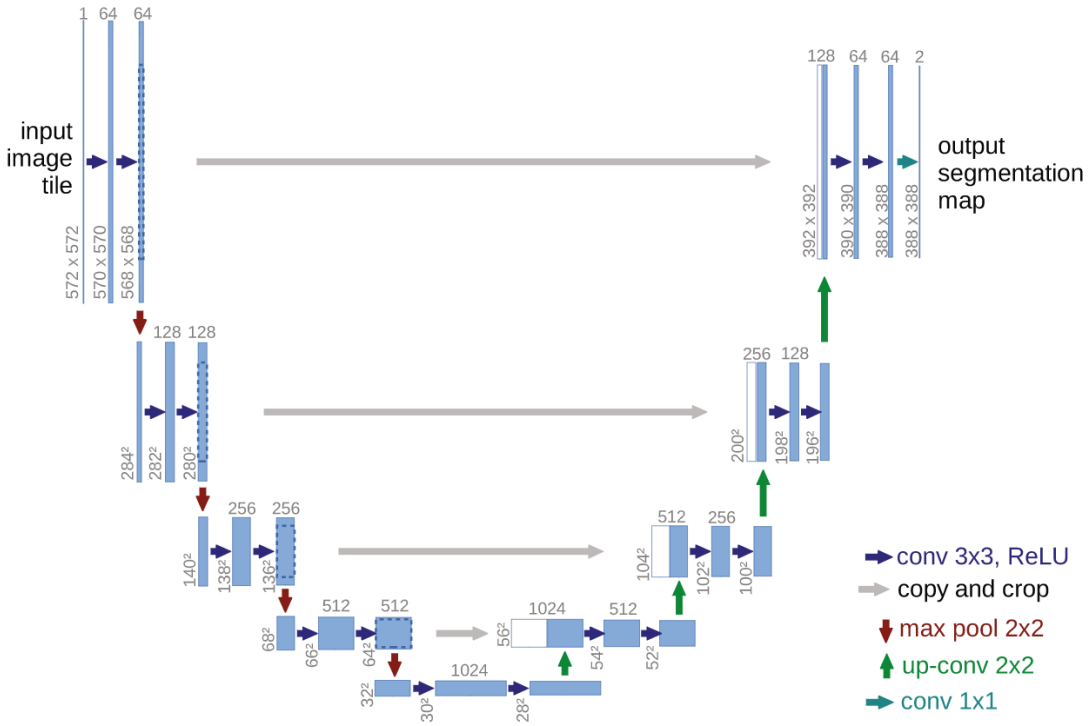

Figure 16. Example of U-Net architecture.

Each blue rectangular represents a multi-channel feature map. On the top of each rectangle, the number of channels is displayed. The numbers at the lower left edge of the rectangles provide the corresponding x-y size of the tensor dimension. Copied feature maps are shown in white rectangles. The arrows denote the operation orientations.

U-Net architecture can be segregated into two components: The down-sampling section, where the x-y size of the tensor dimensions is diminishing, is referred to as the encoder, while the up-sampling section, where the x-y size of the tensor dimensions is enlarging is denoted as the decoder. In deep learning semantic segmentation, it is common to have a large number of network layers, which may generate the problem of gradient vanishing; this can make the training of early layers extremely slow, as well as diminishing overfitted training results [18]. To avoid this problem, a residual neural network (ResNet) was adopted as the structure of the

convolutional neural network, as the gradients in ResNet architecture use skip-connections to remain more correlated [19]. To achieve the required microtubule component location and segmentation in this case, ResNet34 was utilized as the architecture of the encoder, with the decoder set symmetrically.

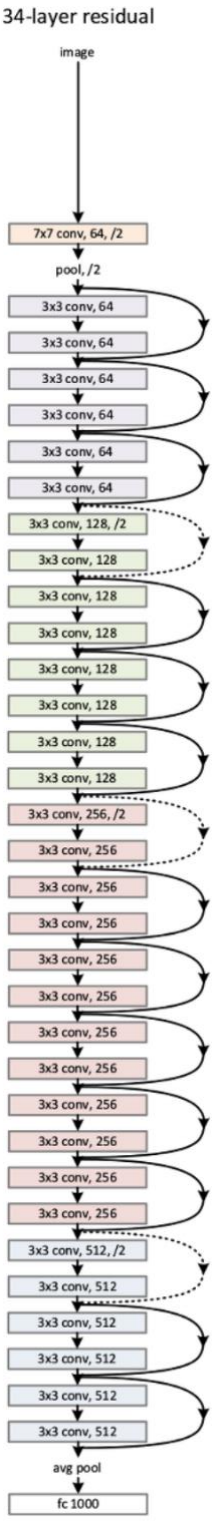

Figure 17. Illustration of ResNet34 network architecture [20].

Precision of the models is critical during the semantic segmentation process in order to derive the four objective parameters accurately. In most circumstances, neural networks utilizing weights from a network pre-trained on a large data set perform better than those trained from scratch on a new dataset [21]. To improve the performance of the U-Net type network in this case, transfer learning was adopted, with ImageNet employed for initial encoder weights.

Adequate selection of loss function is also crucial to the training process of neural networks. In the case of MS and DM detection, these features are usually scattered across the image and only occupy minor regions of that image; this means that the majority of pixel points in the image belong to the background. This means that a binary cross-entropy loss function is not suitable, as this loss function could falsely judge the training accuracy to be high despite the fact that there many MS and DM may not be detected. To correctly reflect accuracy in each training epoch, the Jaccard loss function was thus deployed in relation to the Jaccard index, using the Intersection over Union (IoU) score as a metric.

The Jaccard index, commonly known as the Jaccard similarity coefficient, is a statistic for determining sample set similarity and variety. This is defined as the size of the intersection divided by the size of the union of the sample sets, giving measure of similarity between data sets.

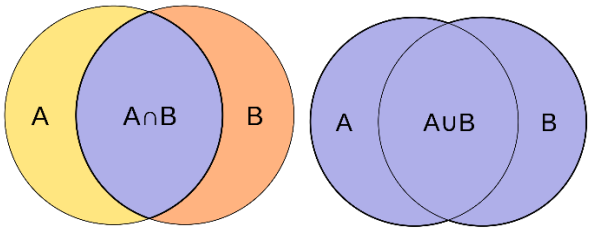

Figure 18. Demonstration of Jaccard coefficient.

The lavender area in the left sub-image represents the intersection between sets A and B, and the lavender region in the right sub-image shows the union between sets A and B.

Given data set A and data set B as shown in the schematic representation in Figure 18, the mathematical formula for the Jaccard index J can be written as

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \quad with\ domain \quad 0 \leq J(A,B) \leq 1$$

Where A and B are both empty sets, the value $J(A,B) = 1$ must be manually assigned.

From the perspective of computer imaging, set A can be considered to be predicted segmentation and set B to be ground truth segmentation; the Jaccard index can thus be expressed as the intersection area between the prediction and the ground truth divided by their union, based on the definition of the IoU metric (Figure 19).

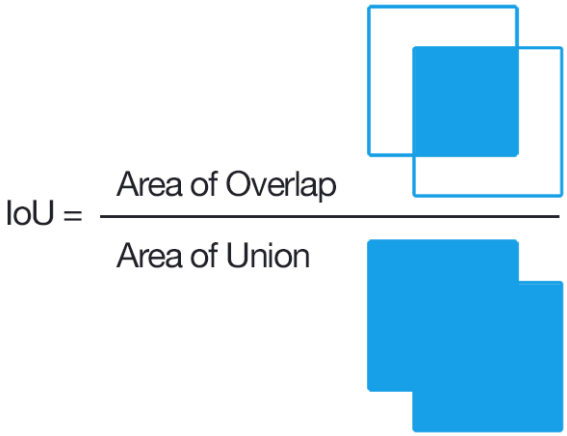

Figure 19. Illustration of the IoU metric.

The upper left square is the prediction and the bottom right square represents the ground truth: the blue block above the line is thus the area of overlap between prediction and ground truth, while the blue area below represents the area of union.

Applying the Jaccard loss function with the IoU metric thus offers a more authentic description of the training accuracy during the training process of semantic segmentation neural networks.

5.2 Tubulin segmentation approach model

To represent the lengths of and to obtain endpoint information regarding tubulins, once the segmentations were obtained, a box model was established, implementing the OpenCV function cv2.minAreaRect(); the segmentation data for each MS and DM were thus individually modularized into minimal area rectangles boxing the segmentations. The two midpoints of both short edges of the rectangles were then assumed to represent the endpoints of the tubulin, and the Euclidean distance between the endpoints was calculated to give tubulin segmentation length based on a measurement unit of pixels.

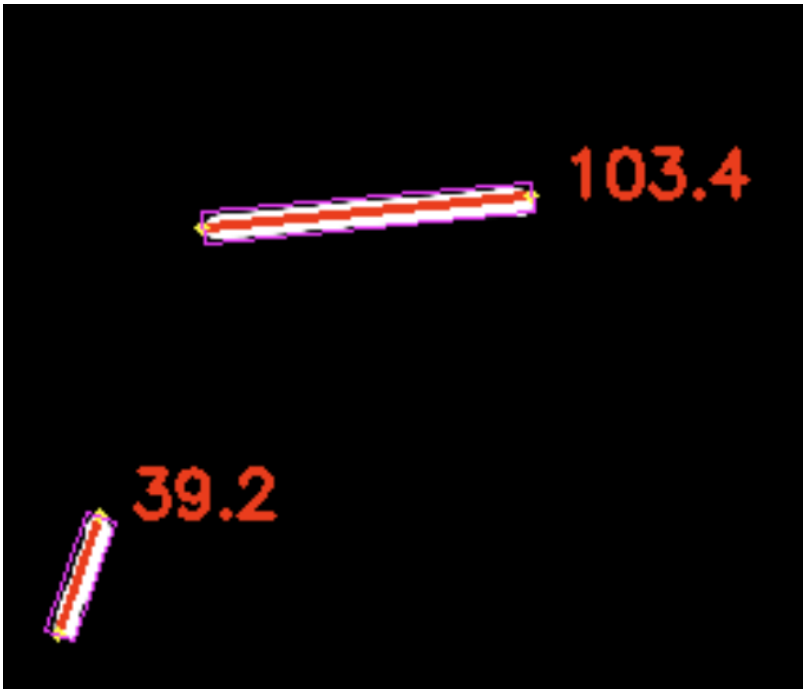

Figure 20. Example of tubulin modularization.

In Figure 20, the white objects are predicted tubulin segmentations, while the violet rectangles that box these segmentations are the results from the OpenCV function cv2.minAreaRect; the yellow points thus represent the midpoints of the short edges of the boxing rectangles, and can thus be taken as the endpoints of the respective tubulins; the Euclidean distances between the endpoints, represented as the red lines, are computed as representations of the length of the corresponding tubulins, with specific lengths figures shown in red numbers. A vibration problem emerged at this stage, however, leading to this the length representation form being replaced with a more precise one later in this work. This length representation form was thus utilized only for detection evaluation.

This model was iterated through all tubulin segmentations in an image to achieve the length information and endpoints coordinates for all detected MSs and DMs.

5.3 Concatenation algorithm

After the decomposition of a DM video into a certain number of DM frame images, the consistency of DM tracking must be strengthened by the addition of endpoint information from the corresponding MSs. To effectively track the DMs, a concatenation algorithm between MS segmentations and DM segmentations was thus developed.

Ideally, during this process, the endpoint of each DM would be assumed to be coincidental with the corresponding MS; based on this, using MS endpoint information as a baseline to

identify the concatenating DM could be an ideal method to successively track DMs. However, as shown in Figure 21, in most cases the endpoints of the MSs do not perfectly overlap with the endpoints of their corresponding DMs, due to the curved shape of DMs, which leads to vibration in recorded DM endpoints. To avoid this problem, an improvement method was adopted to develop more robust results: instead of recording the length for each DM directly, the Euclidean distance between the MS concatenating endpoints and the corresponding DM endpoints at the other end was calculated, and this was used to represent the length of the DM.

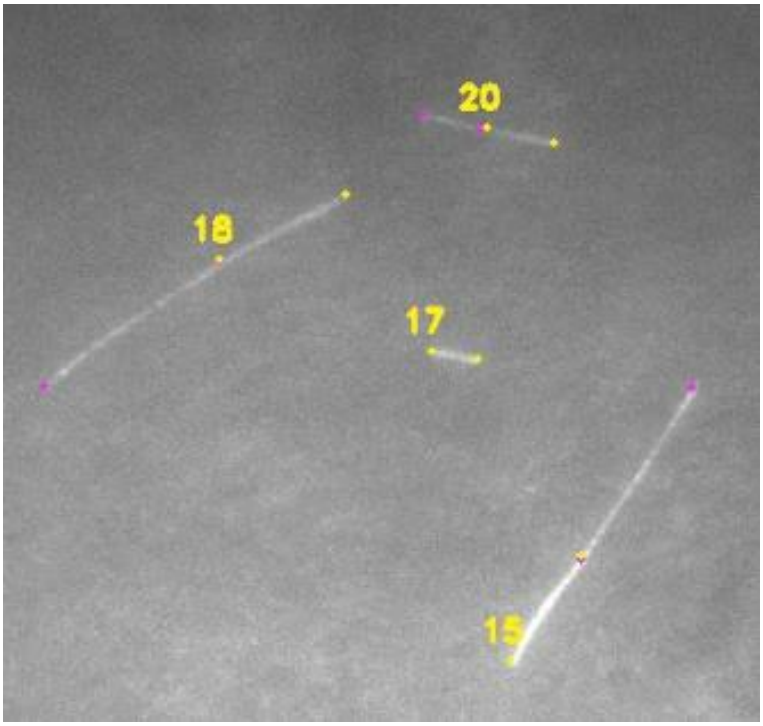

Figure 21. MS and DM concatenation.

Line-shape fragments with yellow dots in the ends are MSs, the yellow numbers represent the serial marker of seeds, and line-shape fragments with violet dots at the ends are DMs.

Once the endpoints of MSs and DMs of all frames were obtained, the concatenation mechanism was implemented. This can be expressed as follows: Given a MS with its endpoints, a DM is assumed to be concatenated to this MS if it satisfies the condition that one of the endpoints of the DM is within the radius of a certain number of pixels of one of the endpoints of the MS; the length of that DM is then documented as the Euclidean distance between the concatenation endpoint of the MS and the other endpoint of the DM.

For a single video frame image, given the set of MS endpoints, the set of DM endpoints, and the threshold radius R, the concatenation algorithm steps are as follows:

1. Sort the MSs and their endpoint pairs from left to right in the MS image.
2. Sort the DMs and their endpoint pairs from left to right in the DM video frame image.
3. Consider the MS endpoints pair SEP_1, SEP_2, and any one of the DM endpoints pair MEP_1, MEP_2: four distances should then be computed: distance $D_1 = \|SEP_1 - MEP_1\|_2$, distance $D_2 = \|SEP_1 - MEP_2\|_2$, distance $D_3 = \|SEP_2 - MEP_1\|_2$, and distance $D_4 = \|SEP_2 - MEP_2\|_2$.
4. For one MS endpoint pair, iterate through all DM endpoint pairs; if any DMs fit the judgment condition $D_1 \leq R$, that DM length should be marked down as D_2 and attached to the corresponding MS; if any DMs fit the judgment condition $D_2 \leq R$, the DM length should be marked down as D_1 and attached to the corresponding MS; if any DMs fit the judgment condition $D_3 \leq R$, the DM length should be marked down as D_4 and attached to the corresponding MS (Figure 22); if any DMs fit the judgment condition $D_4 \leq R$, the DM length should be marked down as D_3 and attached to the corresponding MS. Otherwise, the DM length should be marked as -1 and attached to the corresponding MS.

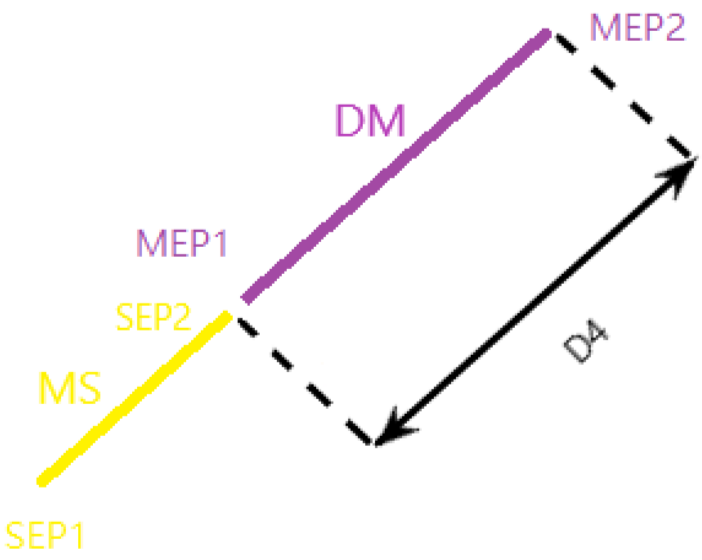

Figure 22. Demonstration of length calculation.

As distance $D_3 = \|SEP_2 - MEP_1\|_2 \leq R$, in this case, $D_4 = \|SEP_2 - MEP_2\|_2$ will be marked down as the DM length.

5. In step 4, if more than one DM exists that meets the judgment criteria, the DM with a longer length will be recorded and attached to the corresponding MS.
6. Repeat steps 3 to 5 for all seeds.

Algorithm 2 Concatenation algorithm

Input: MSendpointsPairList, DMendpointsPairList, R

```
MSendpointsPairList.sort()
DMendpointsPairList.sort()
LengthList ← empty list
for each MSendpointsPair in MSendpointsPairList do
    ConcatenationList ← empty list
    for each DMendpointsPair in DMendpointsPairList do
        D1 ← || MSendpointsPair[0] - DMendpointsPair[0] ||2
        D2 ← || MSendpointsPair[0] - DMendpointsPair[1] ||2
        D3 ← || MSendpointsPair[1] - DMendpointsPair[0] ||2
        D4 ← || MSendpointsPair[1] - DMendpointsPair[1] ||2
        if D1 < R then
            ConcatenationList.append(D2)
        else if D2 < R then
            ConcatenationList.append(D1)
        else if D3 < R then
            ConcatenationList.append(D4)
        else if D4 < R then
            ConcatenationList.append(D3)
        else if D1 < R and D3 < R then
            ConcatenationList.append(max(D2, D4))
        else if D2 < R and D3 < R then
            ConcatenationList.append(max(D1, D4))
        else if D1 < R and D4 < R then
            ConcatenationList.append(max(D2, D3))
        else if D2 < R and D4 < R then
            ConcatenationList.append(max(D1, D3))
        else
            ConcatenationList.append(−1)
        end if
    end for
    if all elements in ConcatenationList = −1 then
        Length ← -1
        LengthList.append(Length)
    else if one element in ConcatenationList ≠ −1 then
        Length ← value of the element
        LengthList.append(Length)
    end if
end for
```

Output: LengthList

Figure 23. Pseudocode of the concatenation algorithm.

Inputs: MSendpointsPairList is the list that containing information about all pairs of MS endpoints, DMendpointsPairList is the list that documented information of all pairs of DM endpoints, R is the threshold radius for the concatenation of MS and corresponding DM.

The implementation of the concatenation algorithm allows a CSV file to be generated in which the DMs length information is noted column-wise for each corresponding MS.

5.4 Resolving overlapping problems

The phenomenon of overlapping tubulins occasionally occurs in both MS images and DM frame images. These overlapping tubulins could appear to be one larger tubulin, be crisscrossed at an arbitrary angle, or be virtually fully on top of one another. It is thus a challenge to distinguish these overlapping segmentations and to find their endpoints.

5.4.1 Check overlapping

To check for overlapping problems after tubulin segmentations were generated, the OpenCV function cv2.approxPolyDP was deployed to approximate each segmentation either with a curve (two vertices) or a polygon (more than two vertices). If the approximation geometry utilized a curve with only two vertices, this was taken to indicate that segmentation had been successful in terms of a single line-shaped tubulin without overlapping issues; however, if the approximation geometry was a polygon with more than two vertices, such segmentation was deemed likely to be composed of two or more overlapping tubulin segmentations.

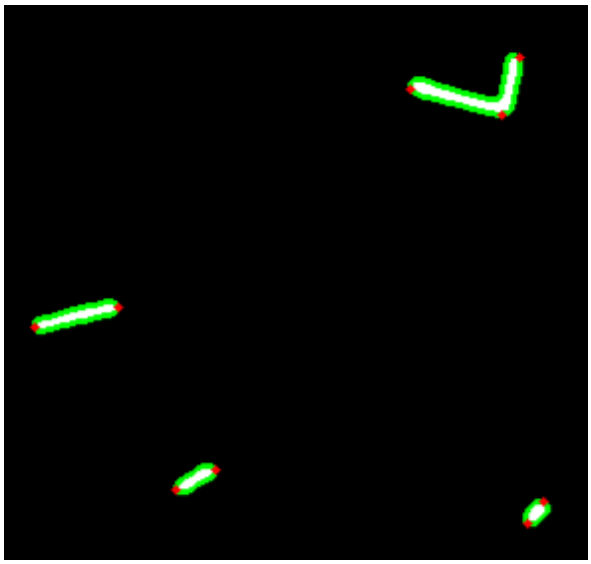

Figure 24. Example of DM segmentation approximation.

The black pixels indicate predicted background, while the white pixels are predicted DM segmentations. The green pixels represent the edges of DM segmentations, while the red dots are the approximation curve/polygon vertices.

Given a binary tubulin segmentation image with background labeled as 0 and tubulin segmentation labeled as 1, overlapping problems can be identified using the following steps:

1. Apply the function cv2.approxPolyDP for one tubulin segmentation, calculating the corresponding approximation curve/polygon and the quantity of its vertices, α.
2. If $\alpha = 2$, this segmentation should have no overlapping issue.
3. If $\alpha \geq 2$, there is an overlapping issue with this segmentation.
4. Iterate steps 1 to 3 through all tubulin segmentations to find all overlapping problems in the image.

There are, however, potentially some limits to searching for overlapping issue using this process; for example, where overlapping tubulins appear to be one larger tubulin, the function cv2.approxPolyDP may still only generate an approximation curve with two vertices, without detecting the overlap at all, which is obviously contradicted. However, most overlapping issues should be spotted in this way.

5.4.2 Overlapping solutions

Empirically, in most cases of overlapping, only two tubulins are overlapped; it is thus readily concluded by an exhaustive enumeration of the overlapping scenarios that the vertices number (α) and the quantity of overlapping tubulins segmentation (β) have the relationship

$$\beta = round\left(\frac{\alpha}{2}\right) \quad with\ \alpha \geq 2$$

To identify which combinations of approximation polygon vertices are the endpoints of overlapping tubulins, a graphs subtraction processing method was adopted. This can be interpreted in the following form: for one tubulin image in a segmentation space, generate a new image that draws a line between any two approximation polygon vertices, calculate the difference between the new image and the original image, record the difference and the index of this pair of vertices, and iterate through all pairs of vertices. Those pairs of vertices with the least difference are regarded as the pairs of endpoints for overlapping tubulins, which then need to be identified.

Given an overlapped tubulin segmentation in a segmentation image where the background is labeled as 0 and the tubulin segmentations are labeled as 1, the endpoints of the overlapped tubulin can be acquired by applying the following steps:

1. Apply the function cv2.approxPolyDP for one tubulin segmentation, calculating the corresponding approximation curve/polygon and its vertices.
2. Assign an empty list named different_pixels_list.

3. Select one pair of the approximation polygon vertices and generate a new image in which a line (3 pixels wide, with a binary value of 1) is drawn between these two vertices: record the index of this pair of vertices.
4. Subtract the new image from the original image to generate a subtraction graph, and count the number of non-zero-pixel points in the subtraction graph; record this number into different_pixels_list.
5. Iterate steps 3 and 4 until all pairs of vertices are calculated.
6. The indexes of first $round(\frac{\alpha}{2})$ smallest in different_pixels_list corresponding pairs of vertices should then be regarded as the pairs of endpoints of the overlapping tubulins and recorded.

Algorithm 3 Overlapping solution

Input: SegmentationImage, SegmentationList, lw

```
EndpointList ← empty list
label = 1
for each Segmentation in SegmentationList do
    vertices ← cv2.approxPolyDP(Segmentation)
    if len(vertices) = 2 then
        there is no overlapping, continue
    else if len(vertices) > 2 then
        VerticesPairsList ← List of all vertices pairs
        DifferentPixelList ← empty list
        for each VerticesPair in VerticesPairsList do
            ProxyImage ← SegmentationImage.copy( )
            cv2.line(ProxyImage,(VerticesPair[0],VerticesPair[1]),label,lw)
            Subimage ← cv2.subtract(ProxyImage, SegmentationImage)
            NonZeroPixel ← cv2.countNonZero(Subimage)
            DifferentPixelList.append(NonZeroPixel)
            P ← round(1/2 len(vertices))
            for the indexes of first P smallest in DifferentPixelList do
                EndpointList.append(VerticesPairsList[index])
            end for
        end for
    end if
end for
```

Output: EndpointList

Figure 25. Pseudocode of the overlapping checking and overlapping solution.

Inputs: SegmentationImage is the input segmentation image of tubulin, SegmentationList is the list that recorded information of all tubulin segmentation, lw stands for the width of the line that needed to be drawn.

Due to the fact that the vertices of approximation polygons are not precisely the endpoints of tubulins, the computed DM length may hold a small inaccuracy; however, its coordinates are close to the actual endpoints, making the error acceptable.

5.6 Regression analysis

5.6.1 Rejecting outliers

A small amount of DM length information may be documented incorrectly due to image noise or missed detections during the segmentation process. Prior to analyzing the DM length information, any anomalous values in the information thus need to be excluded.

To reject outliers for each DM, a 2-step process was performed:

1. All length data values of -1 were rejected initially; and
2. Only data within 3 standard deviations was retained.

5.6.2 Local extreme searching algorithm

After measurement of the length of each DM in each frame, a scatter plot with respect to the frame sequence and the DM length information (/pixels) for each DM was generated, reflecting the variation in DM length over time. In most cases, DMs exhibit both the grow and shrink behaviors, resulting in multiple local extremes in the scatter plot (Figure 26). Occasionally, some DMs only grow or shrink over the course of the video time, leading in no local extreme in the scatter plot (Figure 27).

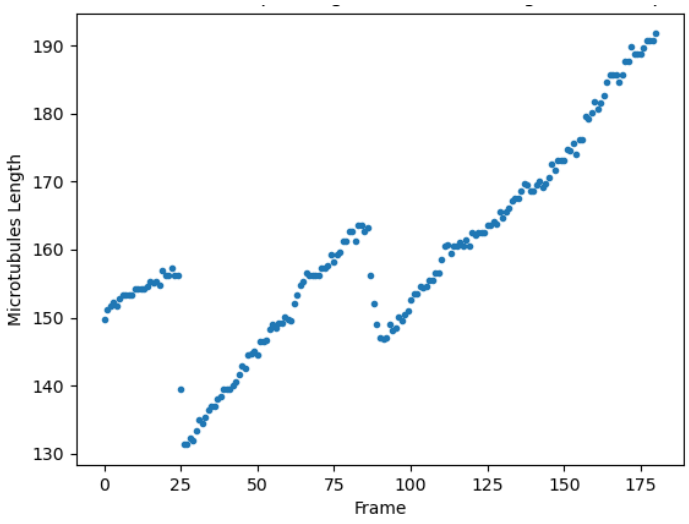

Figure 26. Example of a scatter plot of detection results with mutiple local extremes.

The horizontal axis represents the frame sequence, and the vertical axis represents the corresponding lengths (in pixels) of the DM.

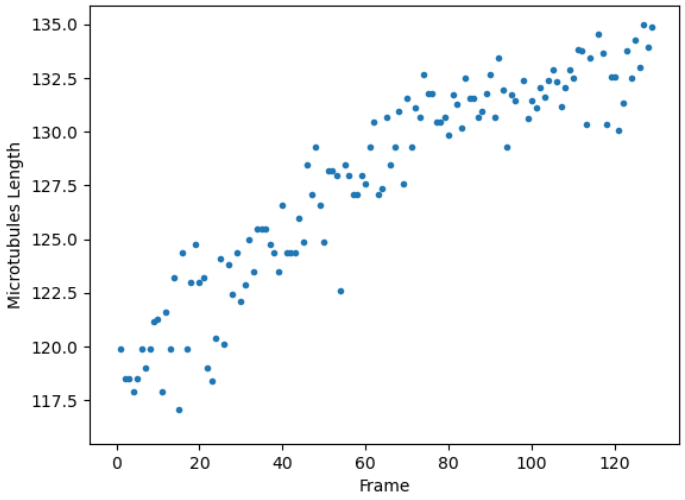

Figure 27. Example of a scatter plot of detection results with no local extreme.

The horizontal axis representing frame sequence, and the vertical axis representing corresponding lengths (pixels) of the DM.

In order to determine the regression analysis method that will be implemented, the local extreme quantity should be known in advance, a local extreme search algorithm was thus formulated such that, where a data value was the largest or smallest amongst a certain number of surrounding data points (in the program, set to 10% of the total data quantity as default), the data point was considered a local extreme.

Given a dataset representing one DM's length information over time, a parameter w (subject to $(2w + 1)$ being smaller than the quantity of the dataset), was set, so that the local extreme search algorithm could follow the following steps:

1. Set a local minimum counter $MIC = 0$ and a local maximum counter $MAC = 0$.
2. Start with the $(w + 1)$th data point, and compare this data value with the W data points before this data point and the W data points after this data point. If the value of this data point is the smallest in this set, mark the data as a local minimal, and set $MIC + 1$; if the value of this data point is the largest, mark the data point as local maxima, and set $MAC + 1$; else, make no change.
3. Iterate through the data along the data sequence, until the last Wth data point.
4. The local extreme quantity then equals $MIC + MAC$.

Algorithm 4 Local extreme searching algorithm

Input: w, Dataset$_l$

```
MIC ← 0
MAC ← 0
MaximalList ← empty list
MinimalList ← empty list
for each data from w-th data to (l − w)-th data in Dataset) do
    LocalList ← empty list
    for each number of the iteration range w do
        PreviousData ← Dataset[data − number]
        LocalList.append(PreviousData)
        FollowedData ← Dataset[data + number]
        LocalList.append(FollowedData)
        if Dataset[data] ≥ max(LocalList) then
            MaximalList.append(data)
            MAC ← MAC +1
        else if Dataset[data] ≤ min(LocalList) then
            MinimalList.append(data)
            MIC ← MIC +1
        end if
    end for
    LocalExtremeQuantity ← MAC + MIC
end for
```

Output: LocalExtremeQuantity, MaximalList, MinimalList

Figure 28. Pseudocode of the local extreme searching algorithm.

Inputs: w is the parameter to define data local, $dataset_l$ is the input dataset which the quantity of l.

Applying the local extreme searching method provides information regarding the local minimum, local maximum, and the number of local extremes.

5.6.3 Piecewise linear regression analysis

In the event of the existence of several local extremes, based on DM length scatter plot morphology and empirical knowledge about DM behaviors, the piecewise linear regression algorithm was deemed suitable for the analysis of the growth and shrinkage velocities of the microtubules.

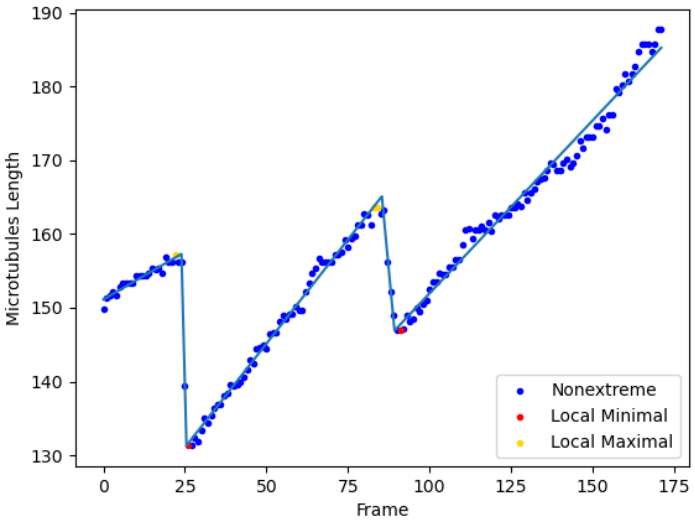

Figure 29. Piecewise linear regression analysis.

Analysis of Figure 26: blue lines represent the linear regressors in corresponding segments.

To implement the piecewise linear regression algorithm, the quantity of breakpoints had to be set manually. The quantity of local extremes can be set as the breakpoint quantity in this case for the implementation.

5.6.4 RANSAC analysis

In some scenarios, several DMs only grow or shrink throughout video recording period, which indicates that the lengths of these DMs increase or decrease monotonically, and there is no local extreme in the scatter plots. The piecewise linear regression analysis is not suitable for such cases, and another regression analysis method is required.

Where the DM length monotonically increases or decreases, yet the value of the first derivatives of the increase or decrease velocity differs at each stage, simply using a linear regression model to obtain velocity values will lead to further inaccuracies or downright errors. As a consequence, the RANSAC algorithm was proposed for use.

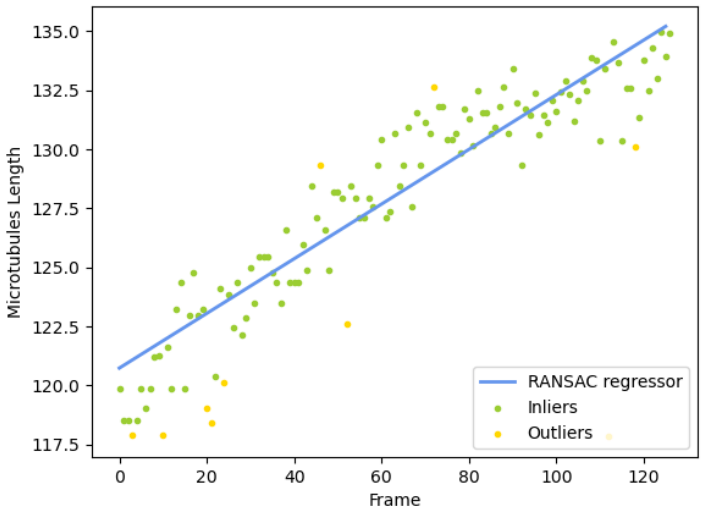

Figure 30. RANSAC analysis.

Analysis results from Figure 27: blue lines represent the RANSAC regressor.

The RANSAC algorithm can address this problem significantly.

6. Implementation

To achieve the gathering of the objective parameters, a general procedure pipeline which automatically located and measured MS and DM was established. The general pipeline proceeds as follows:

1. Load DM video and MS images.
2. Apply image preprocessing.
3. Perform tubulin localization and apply segmentation algorithms.
4. Fit a model of the tubulin to the segmentation results.
5. Strengthen the consistency of DM tracking during the segmentation process.
6. Generate objective parameters.

A general flowchart for this process is given in Figure 31:

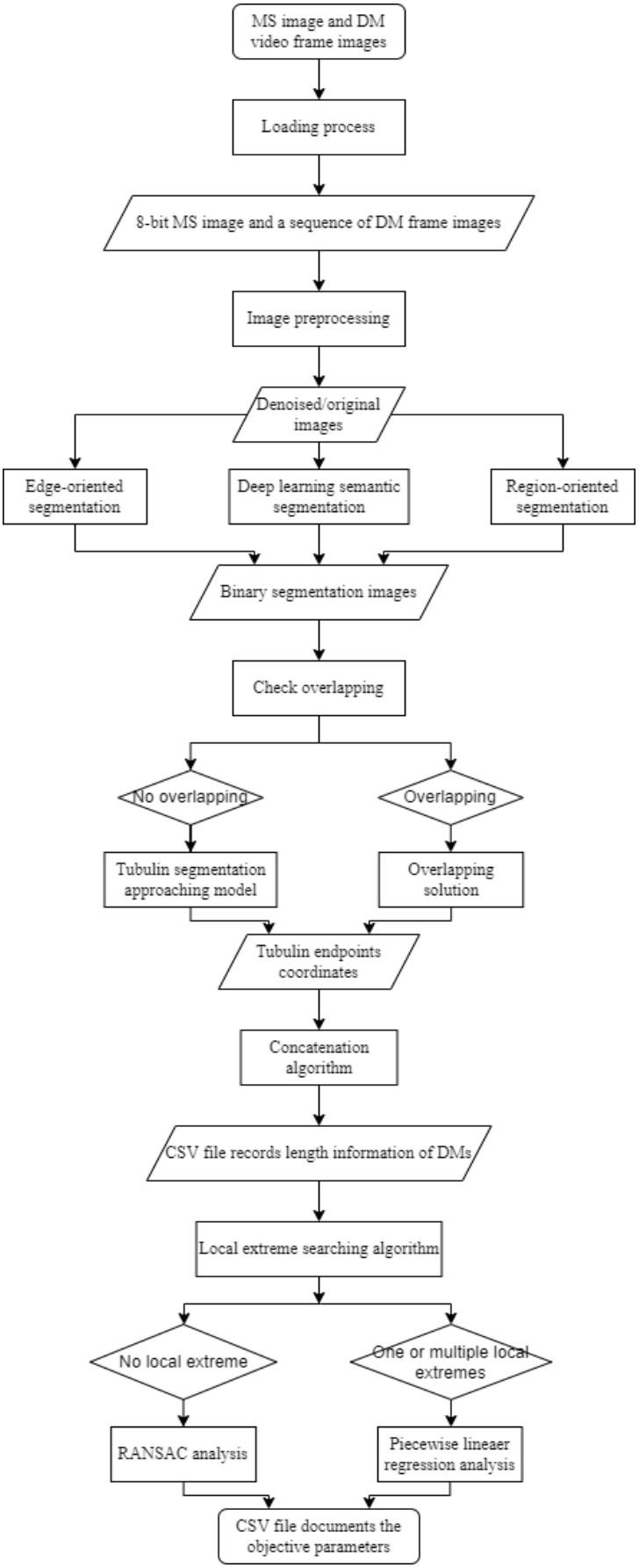

Figure 31. Flowchart of the general implementation pipeline.

The pipeline procedure was, however, adjusted for each relevant segmentation algorithm.

6.1 Edge-oriented segmentation

The processing pipeline for edge-oriented segmentation:

1. Load MS and DM video frame images: Transfer the MS image format from ND2 to PNG and normalize it to 8-bit; decompose the DM video from ND2 format into a sequence of frame images in PNG format and normalize these to 8-bit.
   Input: 16-bit MS images in ND2 format and 16-bit DM videos in ND2 format.
   Output: 8-bit MS images in PNG format and a sequence of DM frame images in PNG format.
2. Image preprocessing: The majority of the image noise is of an additive-white-Gaussian-like noise type; thus, a denoising process with Gaussian kernel convolution filter, also referred to as Gaussian smoothing, is applied to reduce tubulin image noise. The Gaussian kernel size was empirically set to (3, 3) in this work.
   Input: MS image and a corresponding sequence of DM images.
   Output: Denoised tubulin images.
3. Apply edge-oriented segmentation: the Canny edge detector is implemented to locate tubulin images and segment these into binary segmentation images, where a pixel value of 0 represents the background, and a value of 1 indicates a tubulin.
   Input: Denoised tubulin images.
   Output: Binary segmentation images.
4. Check for overlapping and fit a tubulin segmentation approach model: An overlapping procedure check for each tubulin segmentation is necessary, as if the segmentation has an overlapping problem, this must be resolved by applying an overlapping solution. Other segmentations are then used in the tubulin segmentation approach model to gather information about tubulin endpoint coordinates.
   Input: Binary segmentation images.
   Output: Tubulin endpoint coordinates.
5. Implement concatenation algorithm: Once the endpoints of the tubulins are obtained, a concatenation algorithm to match the MS and the corresponding DM in each video frame image is applied to acquire the DM length information. The length information of DMs is then stored in a CSV file organized according to the sequence of MSs.
   Input: Tubulin endpoint coordinates.
   Output: CSV file with DM length information.

6. Generate the objective parameters: The CSV file is read to extract the time-varying length information for each DM; after the rejection of outliers and the implementation of the local extreme searching algorithm, RANSAC or piecewise linear regression can be applied to generate the four objective parameters based on the number of local extremes.
   Input: CSV file with length information of all DMs.
   Output: CSV file documenting the four objective parameters v_g, v_s, f_c, and f_s which reflect the behavior of each DM.

6.2 Region-oriented segmentation

The processing pipeline for region-oriented segmentation:

1. Load MS image and DM video frame images: Transfer the MS image format from ND2 to PNG and normalize it to 8-bit; decompose the DM video from ND2 format into a sequence of frame images in PNG format and normalize these to 8-bit.
   Input: 16-bit MS images in ND2 format and 16-bit DM videos in ND2 format.
   Output: 8-bit MS images in PNG format and a sequence of DM frame images in PNG format.
2. Image preprocessing: The majority of the image noise is of an additive-white-Gaussian-like noise type; thus, a denoising process with Gaussian kernel convolution filter, also referred to as Gaussian smoothing, is applied to reduce tubulin image noise. The Gaussian kernel size was empirically set to (3, 3) in this work.
   Input: MS image and a corresponding sequence of DM images.
   Output: Denoised tubulin images.
3. Apply region-oriented segmentation: The initial point is set as image coordinate (0, 0) in the input image, representing the top-left corner of the DM image; a different coordinate could be used, depending on the specific input image; however, it is imperative that this initial point coordinate is a background point. A region growing algorithm is implemented using an 8-connected neighborhood to segment the background, prior to reversal of the binary value of the background segmentation and the tubulin segmentation. In the segmentation image, a pixel value of 0 thus represents the background, and a value of 1 indicates the tubulins.
   Input: Denoised tubulin images.
   Output: Binary segmentation images.

4. Check for overlapping and fit a tubulin segmentation approach model: An overlapping procedure check for each tubulin segmentation is necessary, as if the segmentation has an overlapping problem, this must be resolved by applying an overlapping solution. Other segmentations are then used in the tubulin segmentation approach model to gather information about tubulin endpoint coordinates.
   Input: Binary segmentation images.
   Output: Tubulin endpoint coordinates.
5. Implement concatenation algorithm: Once the endpoints of the tubulins are obtained, a concatenation algorithm to match the MS and the corresponding DM in each video frame image is applied to acquire the DM length information. The length information of DMs is then stored in a CSV file organized according to the sequence of MSs.
   Input: Tubulin endpoint coordinates.
   Output: CSV file with DM length information.
6. Generate the objective parameters: The CSV file is read to extract the time-varying length information for each DM; after the rejection of outliers and the implementation of the local extreme searching algorithm, RANSAC or piecewise linear regression can be applied to generate the four objective parameters based on the number of local extremes.
   Input: CSV file with length information of all DMs.
   Output: CSV file documenting the four objective parameters v_g, v_s, f_c, and f_s which reflect the behavior of each DM.

6.3 Deep learning semantic segmentation

6.3.1 U-Net network training

Data labeling: The dataset consists of various DM frame images and the MS images, with the latter having different background situations and relatively better image quality; this suggests that a deep learning semantic segmentation neural network trained on data from DM images could offer better predictions for MS images. Thus, as the number of MS images was rather low, it was not considered necessary to include MS images in the training dataset. Among all the DM image data, 26 videos were selected randomly and decomposed into image sequences. In order to avoid data correlation and to ensure no additional bias during the training process, in each sequence, 12 discontinuous images that were distributed as evenly as possible through the whole sequence were chosen as representatives of the image sequences, with the first image of each image sequence being labeled, since the first DM

frame was captured immediately after the MS image, as this was expected to facilitate the concatenation of MS and DM. Thus, a total of 26 image lists with 12 images in each list were made available, for a total of 312 labeled image data points. The selected labeling strategy was somewhat aggressive, which means some blurry DM with ambiguous edges will also be labeled. All labeling processes were conducted using the program “LabelMe”.

Training set, validation set, and test set splits: Of the 26 image lists, five were arbitrarily denoted to be the validation set and another five were set as the test set. The ratio between the training set, validation set, and test set was thus 16:5:5. As image data encountered in use will not be subject to noise reduction pre-processing, the data in the training set, validation set, and the test sets was similarly not subject to denoising procedures.

Image augmentation for training set: For each image in the training set, seven augmented images were generated by applying random combinations of augmentations, including vertical flips, horizontal flips, transposes, random rotations of $90°$, and grid distortion.

Training pre-processing: The U-Net architecture only accepts image sizes of (x, y) where both x and y are divisible by 32. The original tubulin image resolution was (1200, 1200), so the image size was expanded to (1216, 1216) by means of image reflect padding to fit the input requirements of the U-Net neural network. The training dataset was also randomly shuffled to reduce variance and prevent overfitting.

Convolutional neural network architectural: a U-net architecture, with ResNet-34 used as an encoder and ‘ImageNet’ used to generate initial encoder weights to implement the transfer learning training process, was applied. The tensor dimensions for each layer of encoder were (1216, 1216, 3) --- (608, 608, 64) --- (304, 304, 128) --- (152, 152, 256) --- (76, 76, 512) --- (38, 38, 1024).

Training results: A batch size of 2 (limited by GPU VRAM) and a learning rate of 0.00001 (corresponding to batch size) was set, with an early stop criterion of five epochs. The training process actually stopped at 25 training epochs.

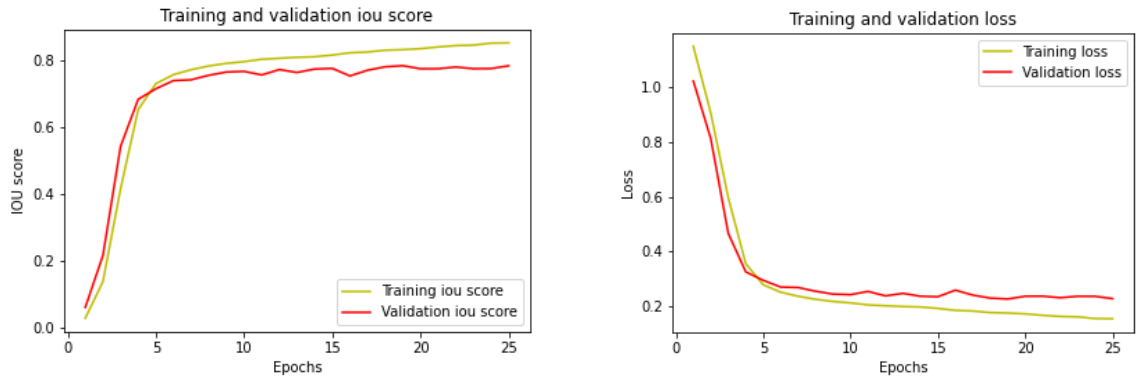

Figure 32. Training and validation scores

The left illustration demonstrates the training and validation IoU score for each training epoch, while the right-hand graph shows the training and validation loss for each training epoch.

The trained network was saved in hdf5 format.

6.3.2 The implementation of semantic segmentation

The processing pipeline for semantic segmentation:

1. Load MS image and DM video frame images: Transfer the MS image format from ND2 to PNG and normalize it to 8-bit; decompose the DM video from ND2 format into a sequence of frame images in PNG format and normalize these to 8-bit.
   Input: 16-bit MS images in ND2 format and 16-bit DM videos in ND2 format.
   Output: 8-bit MS images in PNG format and a sequence of DM frame images in PNG format.
2. Image preprocessing: As no denoising procedure exists during neural network training process, noise is not cancelled at this point.
   Input: 8-bit MS images in PNG format and a sequence of DM frame images in PNG format.
   Output: 8-bit MS images in PNG format and a sequence of DM frame images in PNG format.
3. Apply deep learning semantic segmentation: Input images are padded to (1216, 1216), and a U-Net neural network is used to predict and generate tubulin segmentation images; the output images are then cropped back from (1216, 1216) to the original size (1200, 1200). In the binary segmentation image, a pixel value of 0 represents the background, while a value of 1 indicates a tubulin pixel.
   Input The MS image and the relevant sequence of DM images.
   Output: Binary segmentation images.

4. Check for overlapping and fit a tubulin segmentation approach model: An overlapping procedure check for each tubulin segmentation is necessary, as if the segmentation has an overlapping problem, this must be resolved by applying an overlapping solution. Other segmentations are then used in the tubulin segmentation approach model to gather information about tubulin endpoint coordinates.
   Input: Binary segmentation images.
   Output: Tubulin endpoint coordinates.
5. Implement concatenation algorithm: Once the endpoints of the tubulins are obtained, a concatenation algorithm to match the MS and the corresponding DM in each video frame image is applied to acquire the DM length information. The length information of DMs is then stored in a CSV file organized according to the sequence of MSs.
   Input: Tubulin endpoint coordinates.
   Output: CSV file with DM length information.
6. Generate the objective parameters: The CSV file is read to extract the time-varying length information for each DM; after the rejection of outliers and the implementation of the local extreme searching algorithm, RANSAC or piecewise linear regression can be applied to generate the four objective parameters based on the number of local extremes.
   Input: CSV file with length information of all DMs.
   Output: CSV file documenting the four objective parameters v_g, v_s, f_c, and f_s which reflect the behavior of each microtubule.

7. Evaluation

During the implementation of the appropriate pipeline, two factors primarily influence the objective parameter values: (1) the detection accuracies and recognition capabilities of the segmentation algorithm; and (2) the tracking process used to implement the concatenation algorithm between MS and DM. This evaluation thus focuses on these two factors.

In order to control variables and prevent bias, all evaluation procedures were conducted in the test set. To quantitatively evaluate the precision of tubulin segmentation and the detection abilities of the segmentation algorithms, as well as to assess the efficiency of the concatenation algorithm, it was also necessary to compare the predictions from the programs with a ground truth. The test set data was thus manually labelled to acquire the ground truth, forming a total of five image lists with 12 images in each list. The evaluation data set thus consisted of the 60 DM video frame images, the corresponding 60 ground truth labelled DM

segmentation images, the five relevant MS images, and the corresponding five ground truth labeling MS segmentation images.

7.1 Evaluation of edge-oriented segmentation

The results from the edge-oriented segmentation failed to fulfill expectations. The DM detection suffered from numerous errors and failures:

- The detection algorithm totally failed to identify some DMs.
- A number of DMs were incompletely detected.
- Several DMs were segmented into separate segmentation parts.
- During the detection process, the algorithm falsely recognized a lot of noise as DMs.

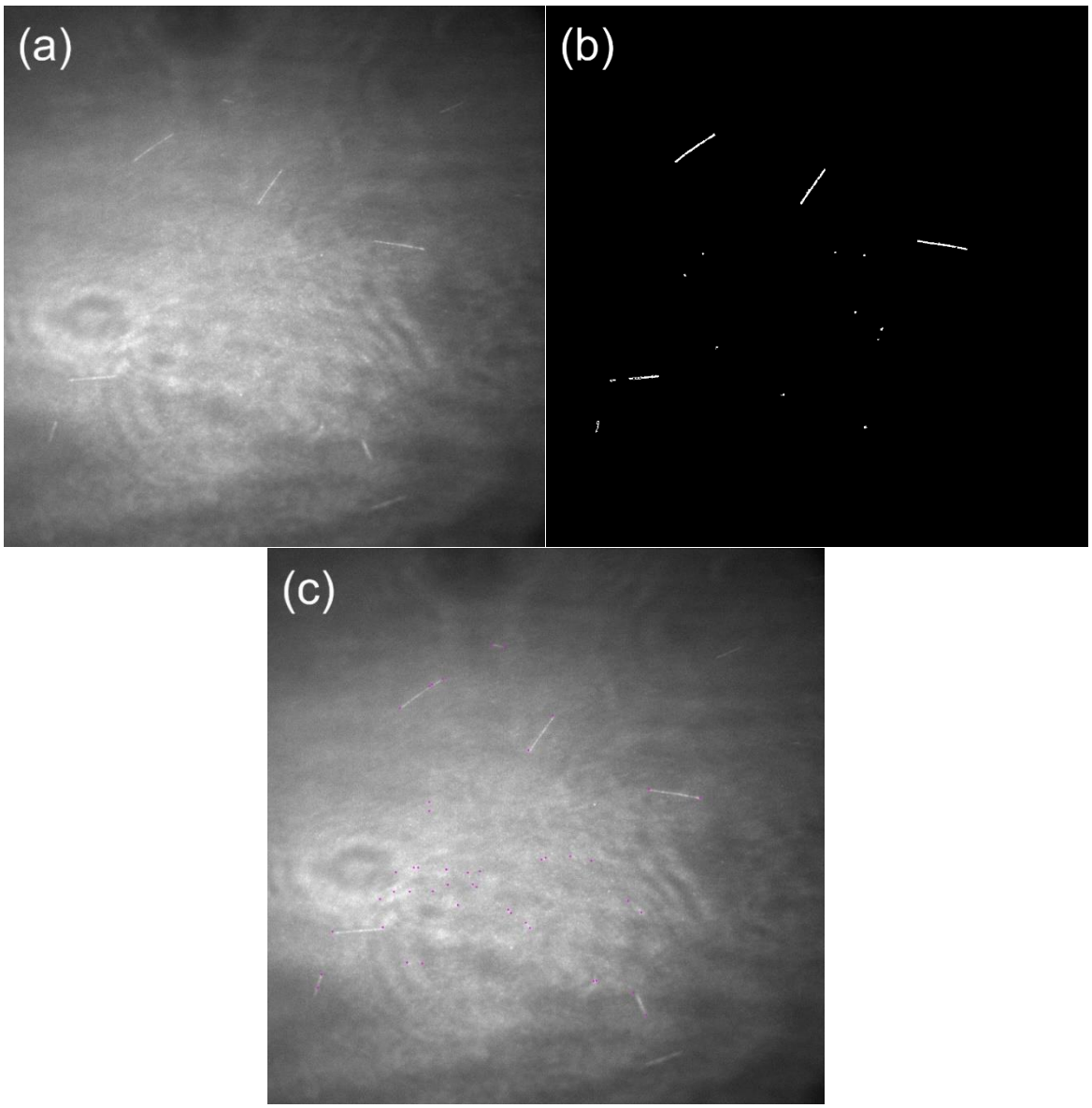

Figure 33. Example edge-oriented segmentation results

Figure 33 (a) is the input image, while (b) represents the output segmentation image. In the segmentation image, many noises are falsely identified as DMs, and a number of microtubules are not detected, while some DMs are detected as separate parts. In image (c), where violet dots show the predicted endpoints of the DMs, as modeled and calculated based on image (b), the endpoints of some DMs have drifted, which means the length predictions will be inaccurate.

There are several potential reasons for these erroneous detection results:

- The contours of some microtubules are very blurred, which causes the edge detection algorithm to fail to recognize the edges of microtubules.
- The illumination during the video was variable, which affected the contrast of several DMs in some frames, reducing the accuracy of edge detection prediction results.
- Most of the decomposed frame images had a large amount of noise with an irregular pattern, and some noise pixels with high gray values were stacked close to certain DMs, which misled the Canny algorithm to segment these as part of these DM.
- Most DMs had small dark regions with significantly lower grayscale values; this phenomenon caused the edge-detection algorithm to assume some DMs were separated parts.

As the detection ability and precision of the edge detection algorithm were not satisfactory, the subsequent steps (pipeline procedure after step 3) were not deemed necessary for further execution and evaluation.

7.2 Evaluation of region-oriented segmentation

In comparison with edge-oriented segmentation, the region-oriented segmentation algorithm provided slightly superior detection capability on average, though with lower detection accuracy. The outcomes from region-oriented segmentation are thus somewhat better than edge-oriented segmentation to some degree, however, they are also unacceptable, and many issues still emerged:

- The region growing algorithm failed to detect several DMs.
- A few DMs were only partially recognized.
- Some DM were detected as divided segments.
- The detection algorithm misidentified a large number of noises as DMs.
- The implementation of the region-growing segmentation algorithm for each DM frame image was time-consuming.

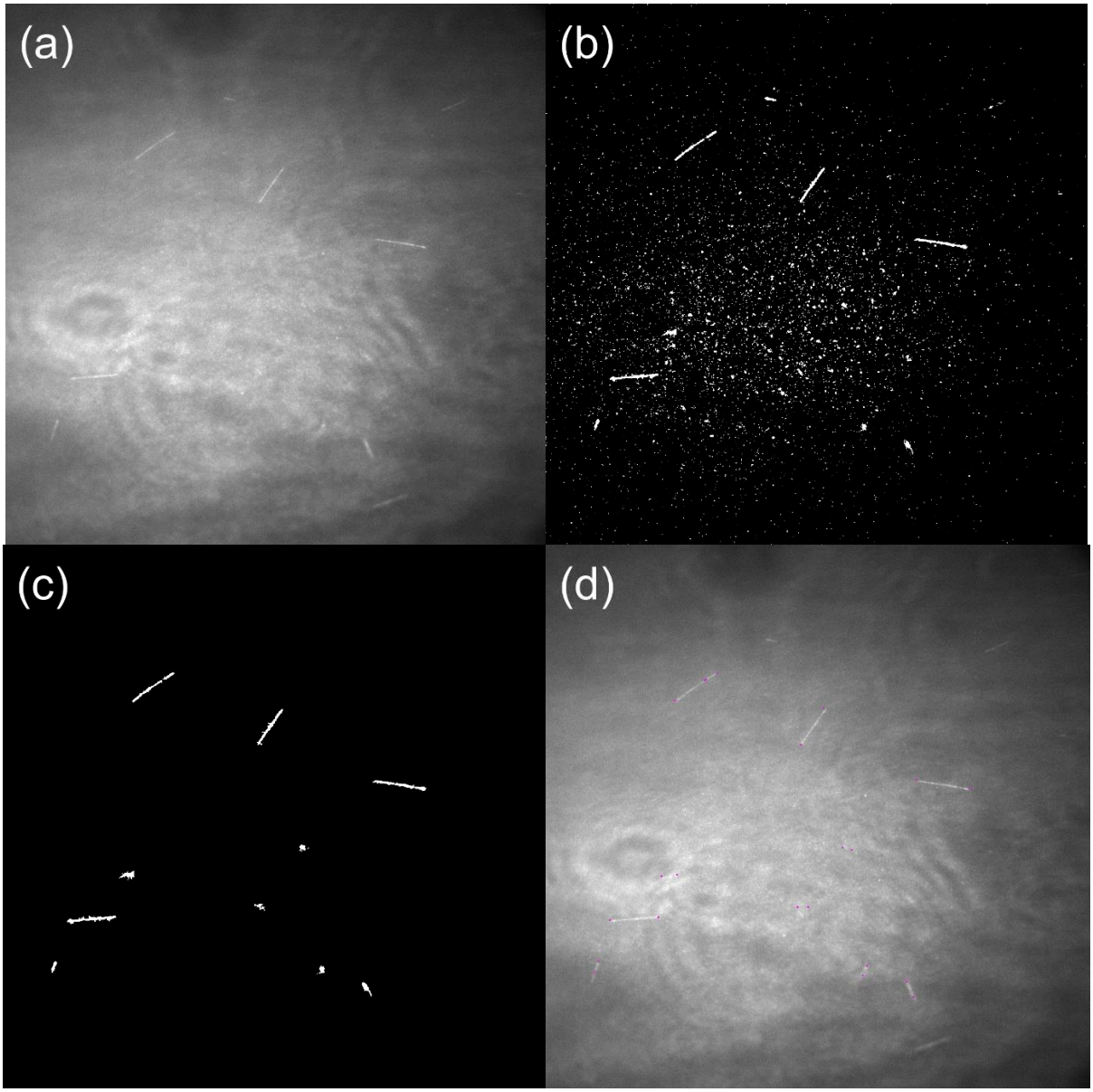

Figure 34. Example region-oriented detection results

In Figure 34, (a) is the input DM image, (b) represents the binary segmentation image, and (c) illustrates the denoised version of image (b). In the denoised segmentation image, while most noises are reduced, a small number of them remain, and several DM are still not detected; further, the phenomenon whereby a few DMs are detected as divided parts remains. In image (d), the violet dots demonstrate the predicted endpoints of the DMs modelized and calculated based on image (c), and several endpoints of DMs are clearly inaccurately located.

There are several potential reasons for these false detection outcomes:

- Several DM are affected by optical blur, and the denoising process could blur the image even more, causing the edges of DMs to have approximately the same

grayscale value as the background; the region growing algorithm will then incorrectly connect the edges of DMs with the background, causing imprecise segmentation of the DMs. If the DMs are sufficiently blurred, the algorithm may totally fail to recognize and segment them.

- A number of DM are very close to noises with almost the same gray values, making it highly probable that the region growing process will attach a DM value to the noises and produce inaccurate predictions.
- When the DM are magnified, many tiny dim segments that have nearly identical gray values as the background can be seen. Those segments are likely to cause separation of the DMs into distinct parts.

The results from region-oriented segmentation also did not satisfy expectations, so the succeeding steps (pipeline procedure after step 3) were deemed unnecessary for further implementation and evaluation.

7.3 Evaluation of deep learning semantic segmentation

Compared to both edge-oriented segmentation and region-oriented segmentation, the deep learning semantic segmentation offered significantly superior outcomes:

- The detection capability of the deep learning semantic segmentation is remarkably effective, with significantly fewer missed detections.
- Deep learning semantic segmentation offers high detection precision, thus measuring the length of DMs more accurately.
- Regardless of the complexity of the noise and poor lighting conditions, deep learning semantic segmentation correctly locates tubulins most of the time, improving the consistency of DM tracking.

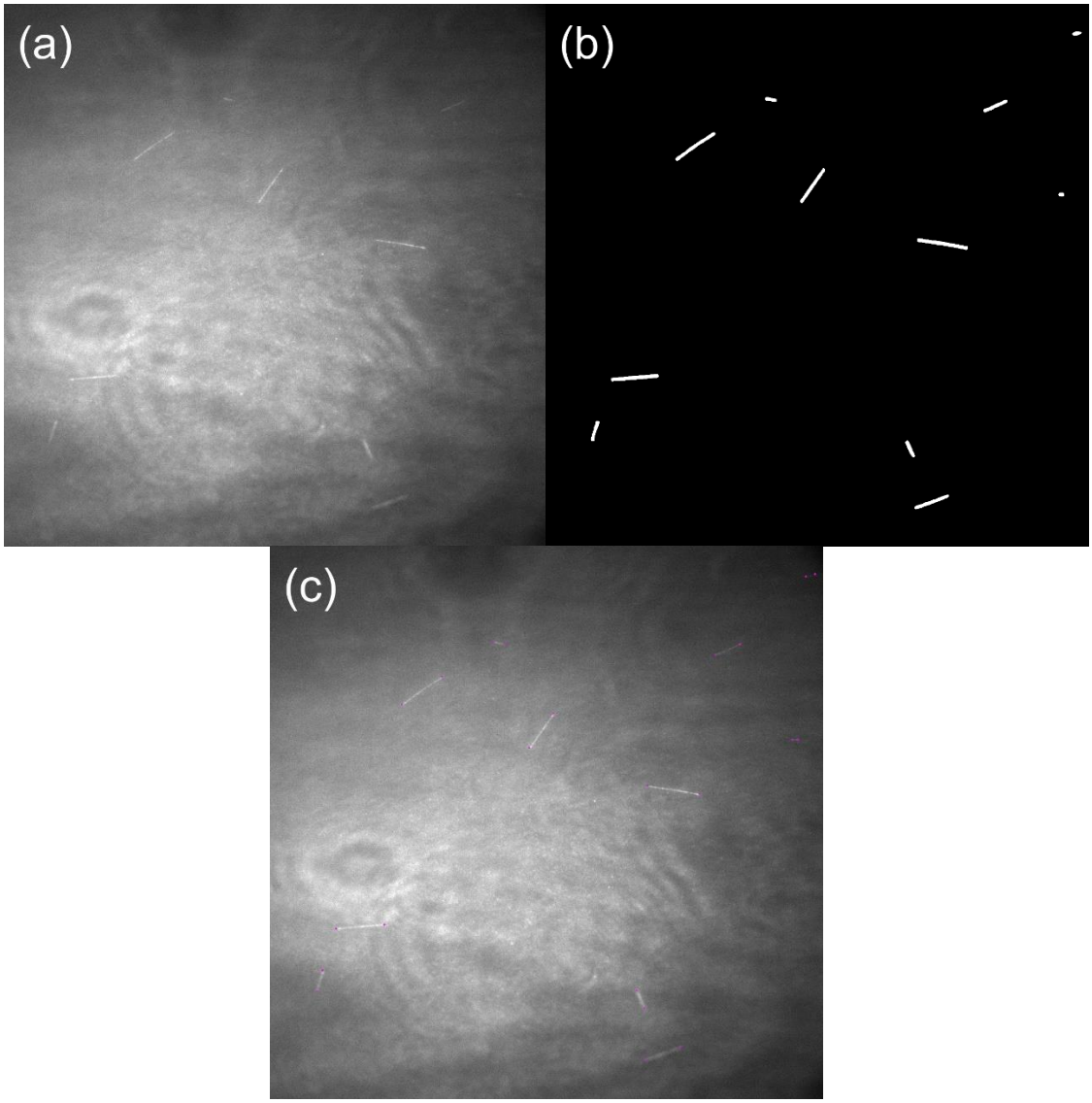

Figure 35. Example deep learning semantic segmentation detection results

Figure 35 (a) is the input image, while image (b) represents the segmentation image, showing that all DM are recognized, though a small number of noises are erroneously identified as DMs; however, these can be rejected during the concatenation process. In image (c), the violet dots show the predicted endpoints of the DMs that were modelized and calculated based on image (b). It can thus be observed that all endpoints from the DM are correctly and precisely predicted in this process.

Despite the excellent performance of deep learning neural networks, they still have the following flaws:

- A few noises are still mistakenly detected as DMs. However, these noises could easily be excluded during the concatenation process with seeds.

- Compared to other segmentation processes, the predicted tubulin segmentation is wider, which might cause derivation of a wider box in the tubulin approach process and thus cause errors in terms of measuring the length of short DMs.
- In some cases, where the frame images are taken with very low illumination, or where the tubulin has ambiguous and blurry edges, the accuracy of detection may be flawed, or detection itself may even be absent.

7.3.1 Evaluation of the trained neural network

To quantify the accuracy of the segmentation procedure, the semantic segmentation procedure was implemented with the trained neural network for only the DM video frame data from the test set, to obtain segmentations. A check to see if overlapping occurred was done, with issues resolved using the standard overlapping solution; where no overlapping, the tubulin segmentation approach model process was applied, and the predicted length information for DMs in each frame generated. To identify the ground truth, repeat the same process to the DM labeled images, without the implementation of the neural network.

In order to match the ground truth segmentations with the corresponding predicted DM segmentations, the centroids of every ground truth segmentation contour and every predicted segmentation contour were calculated; for each ground truth segmentation centroid, if there existed a centroid of a predicted DM segmentation within a Euclidean distance of less than 10 pixels, the predicted DM segmentation was considered to match the ground truth segmentation.

The predicted length information was compared with the ground truth length information to obtain a quantitative accuracy evaluation of the deep learning neural network, the evaluation results were

- Total number of DM segmentation data points: 1,243
- True Positive: 964 (rate: 77.55 %)
- False Positive: 167 (rate: 13.44 %)
- False Negative: 112 (rate: 9.01%)
- The length error larger than 10 pixels: 49 (rate: 3.94%)
- Minimum length error: 0.0 (pixel)
- Maximal length error: 93.27 (pixel)
- Length error mean: 3.30 (pixel)
- Length error variance: 33.15 (pixel2)

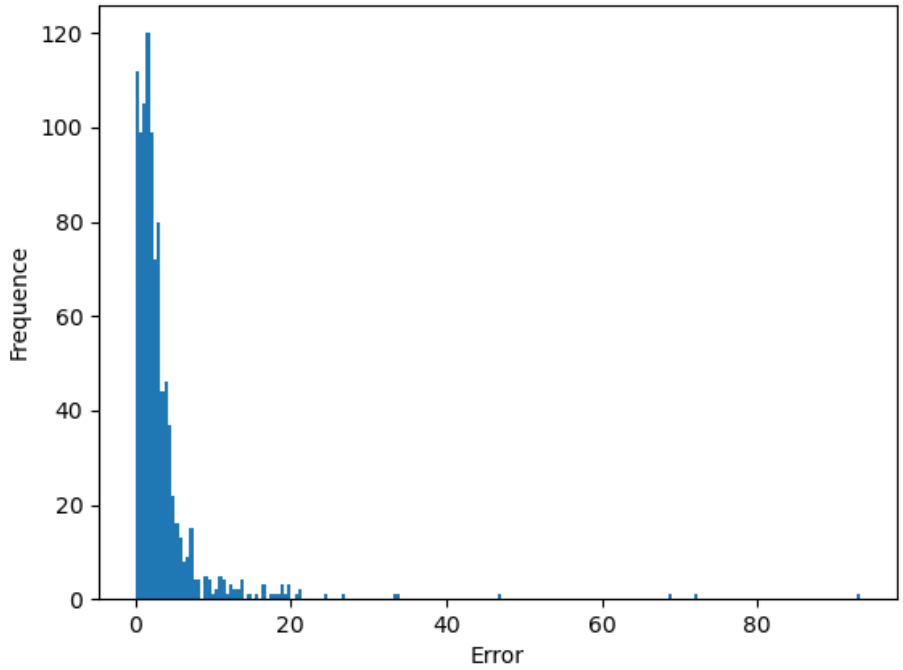

Figure 36. Histogram of predicted DM segmentation length error.

The horizontal axis represents the length error in pixels, while the vertical axis corresponds to the relevant frequency.

In these evaluation results, a false positive occurs where the neural network falsely identifies noise as DM, and the false positive rate is moderately high, however, it is anticipated that the vast majority of noise will be excluded during the concatenation process based on MS information; several circumstances can cause false negatives, though the main ones are (1) misdetection in the neural network and (2) the distance between the centroid of the predicted DM segmentation and the centroid of the corresponding ground truth DM segmentation greater than 10 pixels, resulting in a failed match between prediction and ground truth. However, the relevant histogram reveals that the deep learning semantic segmentation neural network performs exceptionally well, with a relatively small average length prediction error.

7.3.2 Evaluation of the semantic segmentation pipeline

To measure the accuracy of the entire semantic segmentation pipeline, including the concatenation algorithm, for all data in the test set, including DM video frames and MS images, the semantic segmentation pipeline procedure was executed by applying the trained network to generate predicted length information for DMs; to access ground truth data, rather than applying the trained neural network, the labeled DM and MS images were used directly in the semantic segmentation pipeline process. The comparison results were

- Total number of data points: 2,676
- True Positive: 739 (rate: 27.62%)
- True Negative: 1,783 (rate: 66.63%)
- False Positive: 95 (rate: 3.55%)
- False Negative: 68 (rate: 2.54%)
- Minimal length error: 0.0
- Maximum length error: 126.03
- Length error mean: 3.46
- Length error variance: 153.24
- Length errors larger than 10 pixels: 25 (rate: 0.93%)

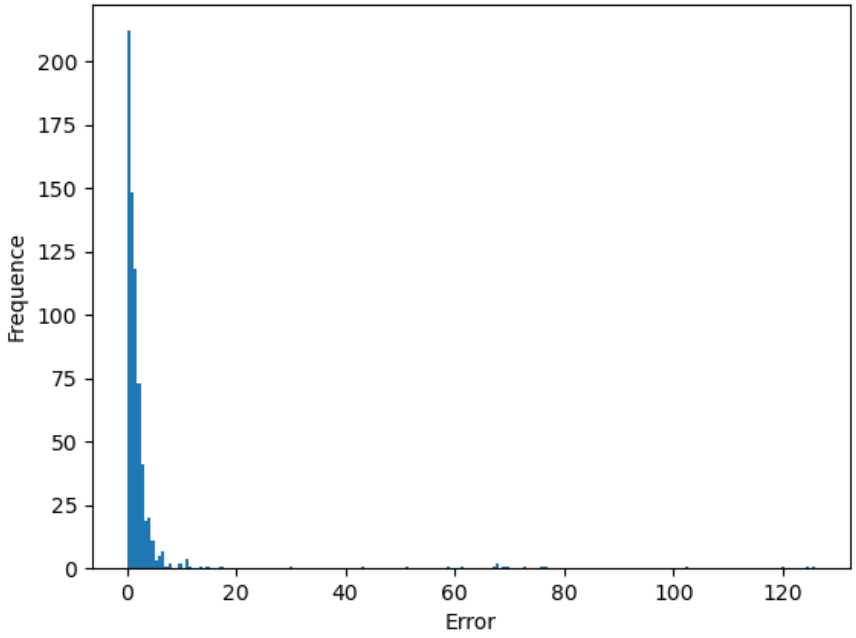

Figure 37. Histogram of predicted DM segmentation length error after concatenation.

The horizontal axis represents the length error in pixels, the vertical axis the corresponding appearance frequency.

For these evaluation results, true negative is used where both the predicted MS endpoints and ground truth MS endpoints are not concatenated; false positive indicates that the predicted MS endpoints were falsely concatenated, whether with white noise or other DM; for false negatives, two possible explanations exist: (1) the neural network missed detecting the DM or (2) the predicted MS endpoints failed to concatenate with the corresponding DM endpoints. In comparison to previous work, however the rates of both false positives and false negatives

were drastically decreased, suggesting both that noise was significantly reduced after the application of the concatenation algorithm, and that the efficiency of the concatenation is acceptable; based on the histogram, it can thus be concluded that the length information for DM as predicted by means of the semantic segmentation procedure is more accurate than the standard approach.

8. Discussion

During the deep learning semantic segmentation procedure, an issue emerged whereby the trained neural network failed to detect, or predicted inaccurate detection of, a tubulin where the tubulin was blurry in the image. Such defects of semantic segmentation by neural networks are likely to occur [22], and while this project lacked the corresponding resources and time to fix this issue, the severity of the problem appeared to be minor. Where a DM suffered from optical blurring, any calculations of length information are likely to be inaccurate; thus, such data should be excluded to ensure better reliability of results. However, if this problem requires resolution for some particular reason, one potential option could be to fine-tune the trained neural network by using blurry training examples over a relatively small number of epochs, as this method can dramatically improve performance on blurry input images [22].

9. Results and Conclusion

Over several months, existing image processing methods were assessed alongside the development of novel models and algorithms to automatically detect and track microtubules with potential overlapping problems. The latter issues were eventually well resolved, and all programs were brought to a functional and usable state. These are now generally available in the GitHub repository at https://github.com/Wirkungstreffer/Microtubules_Detection .

9.1 Automatic tracking and measuring of DM

To automatically extract the four parameters that reflect the behaviors of microtubules, which are the velocity of microtubules growth (v_g), the velocity of microtubules shrinkage (v_s), the velocity at which the microtubules switch from growth to shrinkage (f_c), and the velocity at which the microtubules switch from shrinkage to growth (f_s), the DM recording video must first be decomposed, and the format of the MS image and the DM frame images standardized. The trained semantic segmentation convolutional neural network can then be implemented to generate segmentation images for the relevant MS and DM segments.

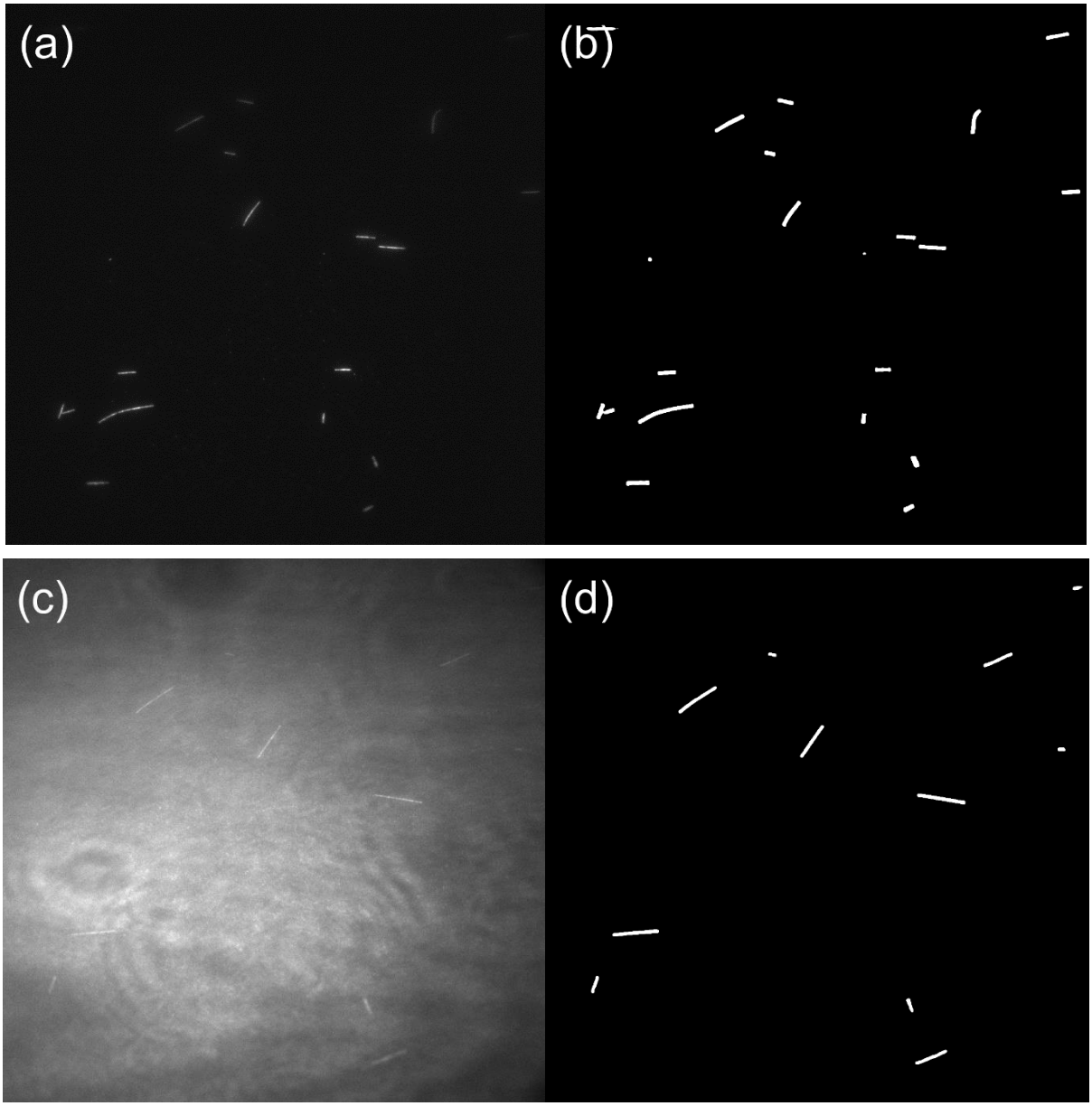

Figure 38. Illustrated examples of deep learning semantic segmentation results

(a) Input MS image; (b) Predicted segmentation image for (a); (c) Decomposed DM video frame image; (d) Corresponding predicted segmentation image for (c).

After checking for overlapping issues for each segmentation, the overlapping solution was applied as appropriate; otherwise, the tubulin segmentation approach model was applied to the segmentations. The endpoint coordinates of the MS and DM in each case were thus acquired using this process.

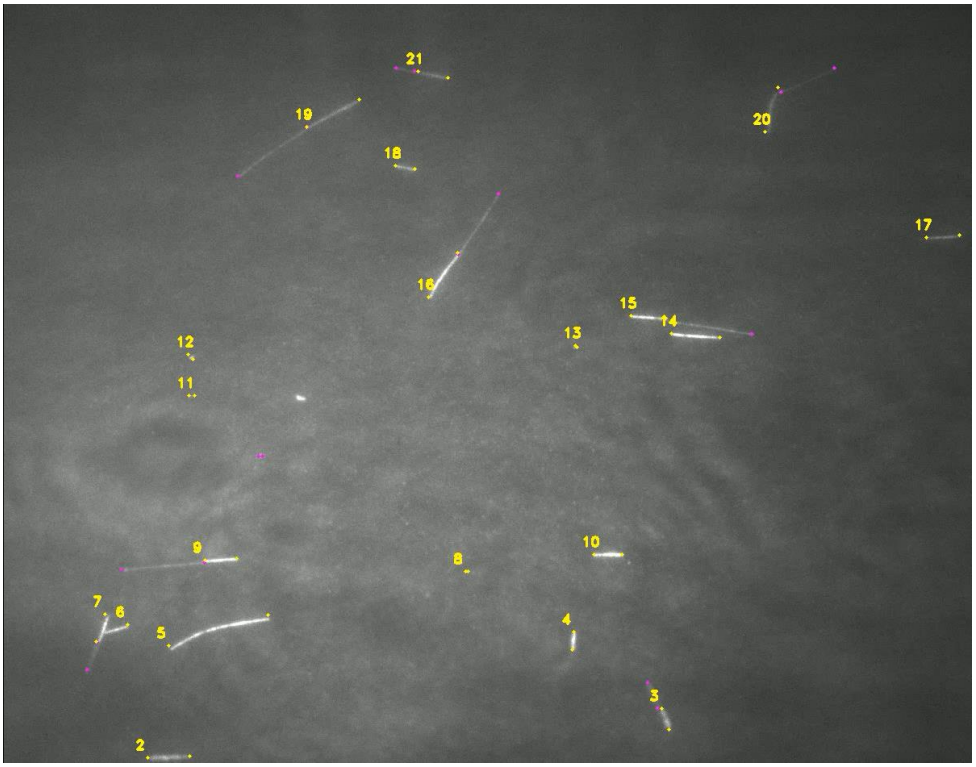

Figure 39. Demonstration of the concatenation between the MS and the DM.

The concatenation results from Figure 38: the line-shaped objects with yellow dots in the endpoints are MSs, with yellow markers giving the serial numbers of the MSs; the line-shaped objects with violet dots in the endpoints are the DMs.

After information about the endpoints was obtained, the concatenation algorithm was applied to link the MS endpoints with their corresponding DM endpoints; the length information for the DM in each frame was then calculated and stored in a CSV file.

The scatter plots of DM length throughout the duration of the video were thus generated based on reading the CSV file.

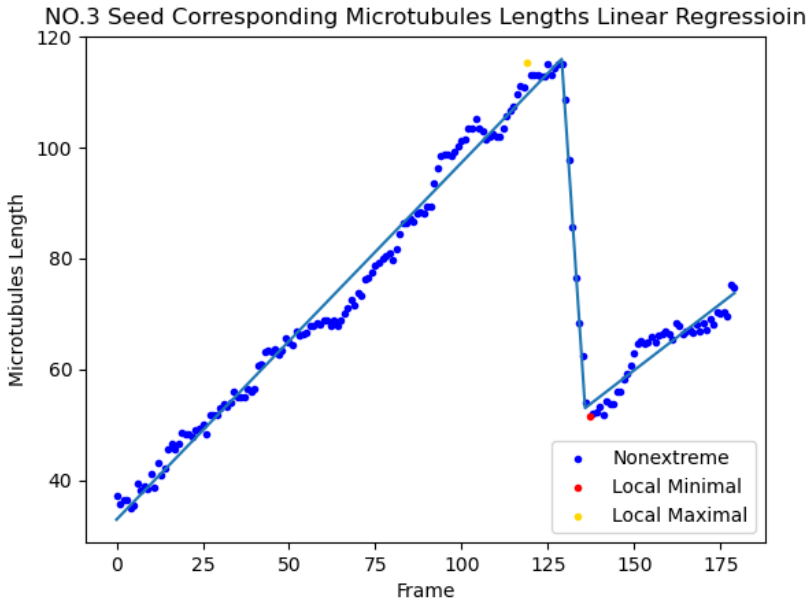

Figure 40. Regression analysis of NO.3 MS corresponding DM.

The horizontal axis represents the frame sequence, while the vertical axis represents the corresponding lengths (pixels) of the microtubule: blue dots are length data, red and gold dots are local extreme length values, and blue lines elucidate the piecewise linear regressors.

In this way, implementation of the RANSAC or piecewise linear regression, dependent on the relevant quantity of local extremes, allows the objective parameters for each DM to be collected. These are then saved in a CSV file.

9.2 Conclusion

By comparing the performance of deep learning semantic segmentation with that of edge-oriented segmentation (e.g., the canny algorithm) and region-oriented segmentation (e.g., region growing), it can be concluded that in the scenario where detected images contain a large amount of additive noise with irregular patterns, deep learning semantic segmentation performs significantly better than traditional image processing methods with respect to locating and measuring small line-shape objects. In fact, it offers considerable advantages not only in terms of detection capability but also detection accuracy in comparison with the other segmentation methods examined.

10 Acknowledgements

Many thanks go to Prof. Dr.-Ing. Olaf Hellwich, Prof. Dr. Guillermo Gallego and Manuel Wöllhaf, who offered detailed professional guidance with respect to the computer visualizations, and to Ella de Gaulejac, who provided all the data and rich and expert descriptions of the biological characteristics of the relevant microtubules. It is also important to acknowledge the Fraunhofer-Institut für Produktionsanlagen und Konstruktionstechnik IPK Berlin and the Fakultät Elektrotechnik und Informatikin for their technical support.

References

[1] Kapoor, V., Hirst, W.G., Hentschel, C. et al. MTrack: Automated Detection, Tracking, and Analysis of Dynamic Microtubules. Sci Rep 9, 3794 (2019).

[2] Brouhard, G. J. & Rice, L. M. Microtubule dynamics: an interplay of biochemistry and mechanics, Nature Reviews Molecular Cell Biology, 1 (2018).

[3] Mitchison, T. Dynamic instability of microtubule growth. Nature 312(5124), 237–242 (1984).

[4] J. Canny, "A Computational Approach to Edge Detection," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986, doi: 10.1109/TPAMI.1986.4767851.

[5] Vision Systems Design, Oct. 1998, p. 20

[6] https://towardsdatascience.com/semantic-segmentation-with-deep-learning-a-guide-and-code-e52fc8958823

[7] https://developer.nvidia.com/discover/convolutional-neural-network

[8] https://www.tobiasbichlmaier.de/semantic-segmentation-with-tf-keras/

[9] Drozdzal, M., Vorontsov, E., Chartrand, G., Kadoury, S. and Pal, C., 2016. The importance of skip connections in biomedical image segmentation. In Deep learning and data labeling for medical applications (pp. 179-187). Springer, Cham.

[10] Freeman, H. and Shapira, R., 1975. Determining the minimum-area encasing rectangle for an arbitrary closed curve. Communications of the ACM, 18(7), pp.409-413.

[11] Eberly, D., 2015. Minimum-area rectangle containing a set of points. Geometric Tools, LLC.

[12] Shamos, M.I., 1978. Computational geometry [Ph. D. Thesis].

[13] Toussaint, G.T., 1983, May. Solving geometric problems with the rotating calipers. In Proc. IEEE Melecon (Vol. 83, p. A10).

[14] Ramer, U., 1972. An iterative procedure for the polygonal approximation of plane curves. Computer graphics and image processing, 1(3), pp.244-256.

[15] Fischler, M.A. and Bolles, R.C., 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, 24(6), pp.381-395.

[16] Golovchenko, N., 2004. Least-squares fit of a continuous piecewise linear function.

[17] Ronneberger, O., Fischer, P. and Brox, T., 2015, October. U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention (pp. 234-241). Springer, Cham.

[18] Hochreiter, S., Bengio, Y., Frasconi, P. and Schmidhuber, J., 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

[19] Balduzzi, D., Frean, M., Leary, L., Lewis, J.P., Ma, K.W.D. and McWilliams, B., 2017, July. The shattered gradients problem: If resnets are the answer, then what is the question?. In International Conference on Machine Learning (pp. 342-350). PMLR.

[20] He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[21] Iglovikov, V. and Shvets, A., 2018. Ternausnet: U-net with vgg11 encoder pre-trained on imagenet for image segmentation. arXiv preprint arXiv:1801.05746.

[22] Vasiljevic, I., Chakrabarti, A. and Shakhnarovich, G., 2016. Examining the impact of blur on recognition by convolutional networks. arXiv preprint arXiv:1611.05760.