Microtubule Detection and Instance Segmentation using Deep Convolutional Neural Networks

by

Tim Holtgrefe

to obtain the degree of Bachelor of Science in Nanobiology
at the Delft University of Technology,
to be defended publicly on Tuesday, 18 July 2023, at 11:00.

Student number:	5310083	
Project duration:	13 February 2023 – 18 July 2023	
Thesis committee:	Dr. ir. I. Smal,	Erasmus MC, supervisor
	Dr. N. J. Galjart,	Erasmus MC

TU Delft   Erasmus MC

Abstract

Microtubules are long cylindrical polymers, assembled from tubulin proteins. Microtubule ends can be visualized using fluorescence and confocal microscopy. This allows for the study of microtubule dynamics. However, the manual annotation of microtubules is laborious, which is why automated tracking methods are used.

In this project we have investigated the automation of both microtubule detection and segmentation using deep convolutional neural networks. To this end, we have adapted the ISOO_{DL} network introduced by Böhm et al. [1]. This network not only detects, but also segments microtubules. Microtubule locations are represented as reference points at their center of mass. Separation between microtubules is introduced by predicting both xz- and yz-sheared masks. At every predicted reference point, post-processing operations revert the shearing operation. This results in microtubule detections and their corresponding segmentations. We have concluded that the ISOO_{DL} network is able to detect and segment microtubules.

In an attempt to further improve the segmentation of overlapping microtubules, we developed a new network to be used exclusively on overlapping microtubules. This network predicts the segmentation mask of every microtubule in a different channel of the output. However, the developed 'overlap network' was unable to produce satisfactory segmentation results. Consequently, this data structure does not facilitate separation between overlapping microtubule.

Acknowledgements

I would like to extend my sincere gratitude to my supervisor, Ihor Smal, for his support throughout this project. I especially wanted to thank him for allowing me to explore what topics I found interesting, while guiding me in the right direction. His approachable nature, and genuine interest in my ideas created an atmosphere where I was comfortable expressing my thoughts.

I would also like to thank Niels Galjart for providing us with the data for this project, and being a member of my assesment committee. I appreciate you taking the time to attend my thesis defence and assessing this report.

I owe thanks to my brother, Niels, for proofreading parts of my thesis and reviewing the many adjustments I made to the figures.

Lastly, I would like to thank my parents, other family members and friends for their emotional support.

Contents

Chapter 1

Introduction

The architecture of cells is governed by a multitude of subcellular structures that orchestrate various biological processes. One of these structures are *microtubules*, long cylindrical polymers assembled from tubulin proteins. A large dynamic array of microtubules, called the mitotic spindle, separates chromosomes and forms the plane of cleavage during cell division. Microtubules act together with actin filaments and intermediate filaments to provide shape and strength to the cytoplasm. Although microtubules play a structural role in the cell, they are highly dynamic in nature. They polymerize, depolymerize and move through the cytoplasm within seconds to minutes [2].

Several proteins have been identified that bind to the ends of growing microtubules. By fusing these proteins with green fluorescent protein, the dynamics of microtubule ends can be studied. Using fluorescence and confocal microscopy, visualizations of this growth can be made in time [3, 4]. However, the manual annotation of microtubule ends is laborious, which is why automated tracking methods are an attractive solution. One advantage of automated tracking methods is the ability to process large volumes of data, resulting in identification of more complex patterns. On top of that automated tracking is reproducible and objective, which allows for standardization of methods between researchers [5].

1.1. Project goal

The majority of automated particle tracking methods follow two steps:

1. *Detection*: Coordinates of microtubules are estimated for every frame in the image sequence.

2. *Linking*: In order to follow microtubules in time, the correspondence between the points across different frames should be established.

In this project we will focus on the detection step of automated particle tracking using deep convolutional neural networks. We will also attempt to segment individual microtubules, which could be useful in the development of a more effective linker. Moreover, segmentation of microtubule ends allows for the analysis of their intensity profiles [6].

1.2. Prior work

Computer vision has improved massively in the past few years, largely due to the success of deep neural networks. In this project, we are interested in *instance segmentation*. Instance segmentation is the classification of individual pixels and assigning them to specific instances of a class. Not only do we want to know where microtubules are, we also want to know which pixels belong to a single microtubule. This is a common task in biomedical imaging, and a lot of work has been done on using deep convolutional neural networks in object detection and segmentation. One example of this is Mask R-CNN, which combines object detection with segmentation of said object [7]. The main network we used, $ISOO_{DL}$ takes a similar region based approach as Mask R-CNN but focuses on overlapping objects [1]. To our knowledge there is no application of deep convolutional neural networks for instance segmentation of microtubules.

1.3. Data

In order to train the networks in this project, a synthetic data generator was used [8]. This generator creates noisy microtubule images, together with their accompanying labels. These labels differ between the networks used, and they are described in the corresponding sections. All synthetic data used in this project has a signal to noise ratio of 5.

In the exploratory phase of this project, we used real annotated microscopy images from Niels Galjart for the training of our networks (see figure 1.1). However, due to the limited amount of data we opted to use the synthetic data to perform our experiments.

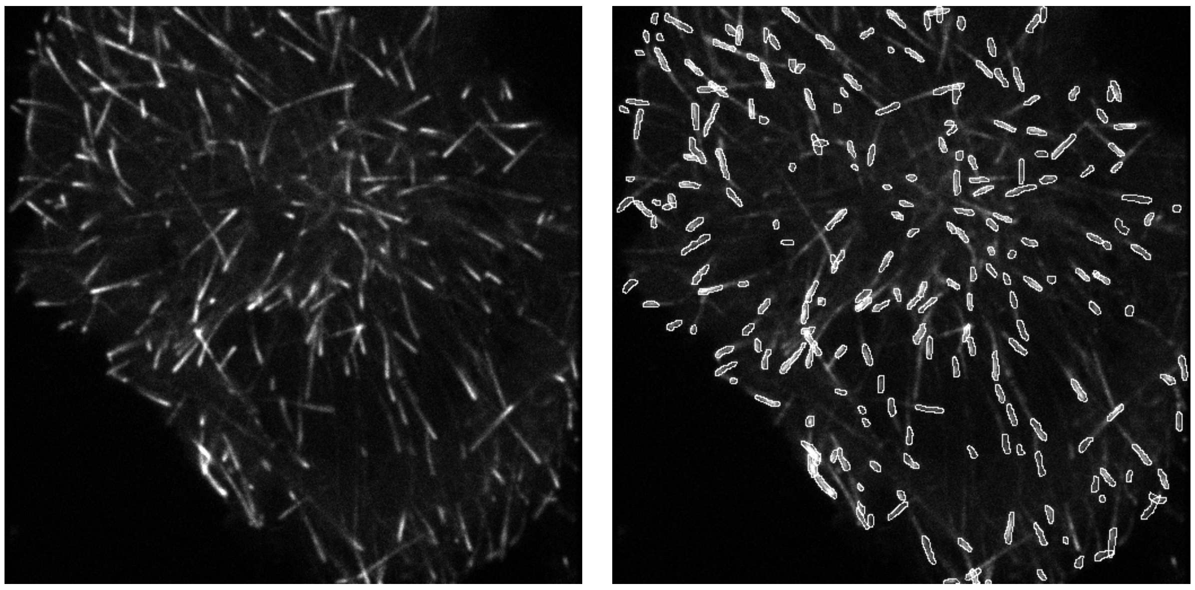

Figure 1.1: An example of how real microscopy images are annotated by biologists.

1.4. Challenges

When tracking microtubule tips, traditional methods such as model fitting and heuristic approaches often encounter common challenges. In this section we go over the challenges that were addressed in this project.

1.4.1. Overlapping microtubules

A problem arises when microtubules are overlapping. Figure 1.2 illustrates how traditional detection methods might struggle with overlapping microtubules. Since the microscope images are limited to 2 dimensions, there is no available information regarding the depth of microtubules. One might expect that some estimate of depth could be made based on the intensity of the microtubule tips: a tip with a higher intensity might lie on top of a tip with a lower intensity. It is important to note that in real data, microtubule tips are sometimes saturated to the highest intensity. This means that if two microtubules overlap, there is almost no change in intensity between the overlapping parts. Not only does this pose a problem for the detection of microtubules but also for instance segmentation. In order for the network to learn this, the microtubules have to be separated in the data. Finding a way to separate microtubules in the data together with a corresponding network that can learn this separation was the main problem faced in this project.

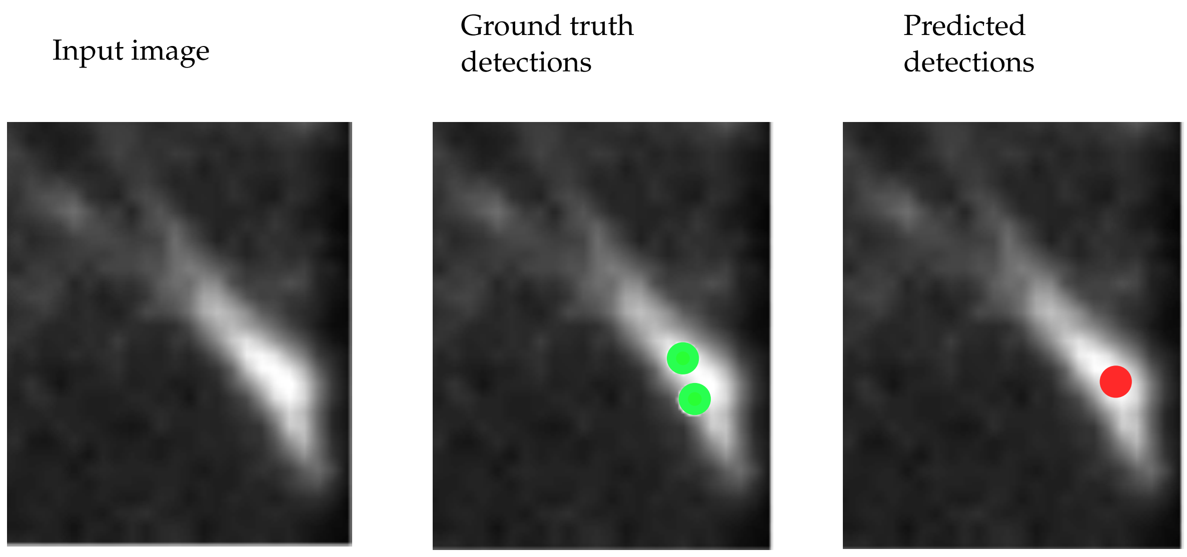

Figure 1.2: An example of how traditional microtubule detection methods might make an incorrect detection in the case of overlapping microtubules.

To achieve these goals, the following experiments were conducted:

1.5. Overview

This project revolved around answering the following research questions:

1. Can we use a deep learning model to automate the detection and segmentation of microtubles?
   (a) Can we *detect* both isolated *and* overlapping microtubules?
   (b) Can we also *segment* isolated *and* overlapping microtubules?
2. Can we improve the segmentation of microtubules using a different network structure?

1.5.1. Experiment 1

The first experiment revolves around the ISOO$_{\mathrm{DL}}$ network structure proposed by Böhm et al. [1]. This network can detect and segment overlapping objects. This network was compared against a regular U-net architecture. Because the amount of real data was limited, a synthetic data generator was used for the training of both networks. The performance of detection and segmentation was evaluated using ROC and FROC curves, as described in section 2.3.

1.5.2. Experiment 2

The second experiment builds upon the results of the first experiment. A second network was trained that aims to improve the segmentation of overlapping microtubules. This network does not have to analyze an entire image but only sections of the image where the

$ISOO_{DL}$ network detects overlap. This network was also trained on synthetic data and aims to segment each microtubule separately in different channels of the output as described in section 2.2.3. The performance of this network was evaluated qualitatively. We have trained the network with 4 different sets of parameters, and will show the resulting segmentations.

1.6. DeepImageJ

The field of deep learning in biomedical image analysis is advancing at a rapid pace. However, it is important to keep in mind that accessibility to these new and improved models is often limited for users with marginal programming experience. Therefore, we have also explored the implementation of neural networks using the deepImageJ environment [9]. The deepImageJ environment allows for easy implementation of neural networks into ImageJ [10]. Members of the bioimaging community tend to have large familiarity with open-source softwares such as ImageJ, which should lead to a more user-friendly experience [11]. The deepImageJ environment is also compatible with using patches instead of full images, and can thus be used on a variety of image sizes. However, the deepImageJ environment is only compatible with pytorch [12] version 1.9.1, while version 2 is already available.

Furthermore, deepImageJ formatted models can be uploaded to the bioimage model zoo [13], a model repository where pre-trained models can be shared between users.

Chapter 2

Method

In this chapter we introduce some basic concepts in deep learning. Additionally, we explore the structures of the networks used in this project. Afterwards we explain the performance metrics used in chapter 3.

2.1. Deep Learning

Deep learning has rapidly become a prominent technique in the field of Artificial Intelligence due to its remarkable ability to learn and model complex relationships within large datasets. Before the advent of deep learning, researchers had to provide their machine learning models with several predefined pieces of information. Each piece of information included in the representation of the data is called a *feature*. The model would take these features and learn which combination of features lead to the desired outcome. The manual selection of features is incredibly time consuming and also difficult. To illustrate this, imagine we would like to detect a face in a photograph. It is highly challenging to describe what a face should look like in terms of pixel values, as there may be shadows falling on the face, or the eyes might be closed. Therefore, newer machine learning models not only learn how to produce an output from features, but they learn to make the features themselves. In this section we will go over some basic concepts in deep learning. For a more extensive overview please refer to [14].

2.1.1. Artificial Neural Networks

The simplest type of artificial neural network (ANN) is a perceptron. A perceptron combines multiple inputs x_i, their weights w_i, and bias b to form output y:

$$y = A(b + \sum_i w_i x_i), \tag{2.1}$$

where A is called the *activation function*. Activation functions are usually non-linear, which enables the network to learn complex non-linear relationships between input and output. More complicated ANNs are made by combining multiple perceptrons and letting the output of one perceptron serve as the input of another perceptron. Neural networks tend to consist of multiple layers, with each layer functioning as a set of perceptrons operating in parallel. An additional layer of perceptrons receives input from the previous layer of perceptrons. A network has an input layer, hidden layers and an output layer (see figure 2.1).

2.1.2. Gradient descent

By manipulating the parameters of the network, we can alter its output for a given input. The optimization of these parameters is what we refer to as the *training* of the network.

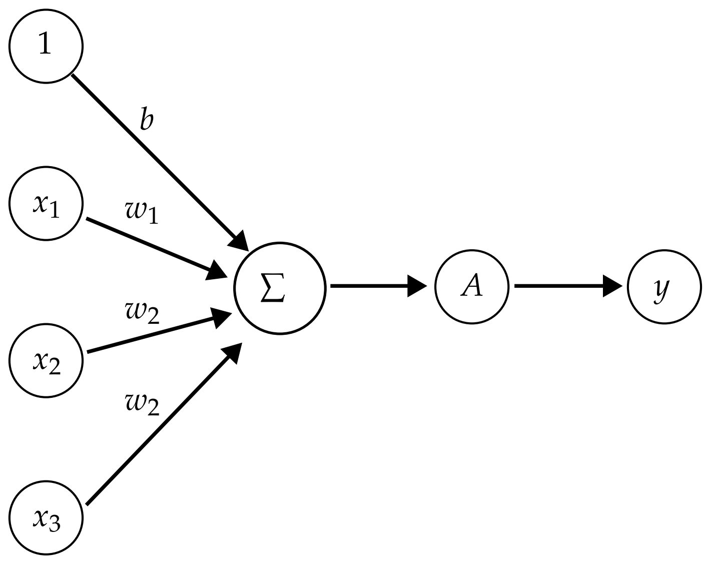

Figure 2.1: A simple perceptron.

The parameters that are modified during the training process are the weights w_i and biases b of the perceptrons. In order to change the values of these parameters a measure of performance is required. Let

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_M \end{bmatrix}, y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}, Y = \begin{bmatrix} Y_1 \\ \vdots \\ Y_N \end{bmatrix} \tag{2.2}$$

be the input, output and expected output of an ANN for *one* training example, respectively. A loss function $f(y, Y) : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$ is defined which is small when our output is close to the expected output and vice versa. During training we present the network with a dataset of input and with expected output. Let

$$\theta = \begin{bmatrix} w_1 \\ \vdots \\ w_k \\ b_1 \\ \vdots \\ b_l \end{bmatrix} \tag{2.3}$$

be a column vector of all trainable parameters in the perceptrons. In order to get a more robust measure of performance for the network we define the cost function $C(\theta)$ as the average loss of all the loss functions for each y_j and the corresponding Y_j. During each iteration of training the cost of the network is computed using the aforementioned cost function. By attempting to minimize this cost the performance can be increased. This optimization is done using gradient descent [15].

Given the cost function $C(\theta)$, gradient descent iteratively updates the parameters by subtracting the gradient of $C(\theta)$ with respect to the parameters θ, multiplied by a learning rate α. Mathematically, the update rule for gradient descent can be expressed as:

$$u_t = -\alpha \nabla C(\theta_t), \tag{2.4}$$

$$\theta_{t+1} = \theta_t - u_t. \tag{2.5}$$

Here, θ_t represents the parameters at iteration t, and $\nabla C(\theta_t)$ is the gradient of the function evaluated at θ_t. The vector u_t is called the *update vector*. The learning rate α controls the step size taken at each iteration. Conceptually this can be understood better by reducing the dimensionality of the cost function to a 3 dimensional surface. The parameters are updated so that the direction of the step is towards the steepest descent in that parameter space. If the step size is too small it will take many iterations to reach a minimum in the parameter space. If the step size is too large the minimum can be overshot. Finding $\nabla C(\theta)$ can be done using the *backpropagation* algorithm [14].

Calculating the gradient for every single training example during each training iteration takes a lot of time. Therefore, it is more common to randomly divide the training examples into smaller batches. The cost and subsequently the gradient of this batch is calculated, and the weights and biases of the model are updated according to this batch. Although this means that the step that is taken in the parameter-space might not be the most efficient, it allows for many more steps to be taken in the same amount of time. The reduction in calculations per batch also allows for using the parallel processing capabilities of GPUs, which speeds up training even more. This method of calculation gradients using a subset of training examples is called *mini-batch gradient descent*.

2.1.3. Optimizers

The traversal of the hyperparameter space is not trivial. Local minima, and more importantly *saddle points*, pose a problem to gradient descent [16]. Saddle points tend to be surrounded by regions with a near-zero gradient. Therefore, they take a very long time to traverse, or they can be seen as a minimum and the training is stopped. In order to traverse the hyperparameter space more efficiently, it is common to use *optimizer algorithms*. Optimizers play a crucial role in training neural networks via gradient descent by efficiently navigating the high-dimensional parameter space and guiding the model towards convergence. The optimizer used in this thesis is *Adaptive Moment Estimation* (Adam) [17]. Adam combines ideas from both momentum-based optimization and adaptive learning rate methods. Momentum-based optimization is a method that increases the gradient vector in the direction of previous gradient vectors (see figure 2.2):

$$u_t = \gamma u_{t-1} + (1-\gamma)\nabla C(\theta_t), \tag{2.6}$$

$$\theta_{t+1} = \theta_t - u_t. \tag{2.7}$$

The hyperparameter $\gamma \in [0, 1)$ is the momentum term that determines how much influence previous update vectors have on the new update vector. Adam took inspiration from the adaptive learning rate algorithm *Root Mean Square Propagation* (RMSProp) [18]. It changes the accumulation of gradient to the exponentially moving average of the squared gradient:

$$v_t = \kappa v_{t-1} + (1-\kappa)\nabla C(\theta_t)^2, \tag{2.8}$$

with $\kappa \in [0, 1)$. Now the update vector u_t is defined as:

$$u_t = u_{t-1} - \frac{\alpha}{\sqrt{v_t + \varepsilon_0}}\nabla C(\theta_t), \tag{2.9}$$

where ε_0 is set as 10^{-8} to prevent division by 0. The idea of RMSProp is that the size of the update vector in one dimension is reduced if it is very large and increased if it is very small. This leads to a straighter path in the hyperparameter space, which helps with convergence.

The Adam optimizer changes the update vector based on the exponential moving averages of the gradient and of the squared gradient, m_t and v_t respectively:

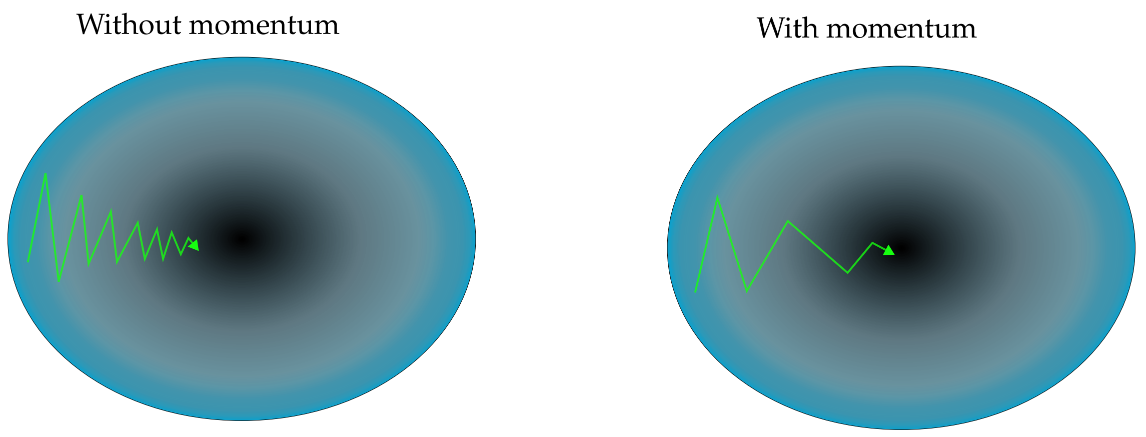

Figure 2.2: Gradient descent with momentum increases the gradient vector in direction of previously taken steps

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla C(\theta_t), \tag{2.10}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\nabla C(\theta_t)^2, \tag{2.11}$$

where $\beta_1, \beta_2 \in [0,1)$ are the hyperparameters governing the exponential decay rates of these moving averages. Commonly β_1 and β_2 are set to 0.9 and 0.999, respectively. m_t and v_t are both initialized to 0, and since β_1 and β_2 are almost 1, m_t and v_t have a bias towards 0. The bias-corrected versions of m_t and v_t are defined as follows:

$$\hat{m}_t = \frac{m_t}{1-\beta_1}, \tag{2.12}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2}. \tag{2.13}$$

The update vector of the Adam algorithm thus becomes:

$$u_t = \alpha \frac{\hat{m}_t}{\sqrt{\hat{v_t}} + \varepsilon_0}. \tag{2.14}$$

2.1.4. Regularizers

The main objective of a trained neural network is to perform well on data it has not seen before. Therefore, the performance of the network on unseen data is measured during training. In order to achieve this the labeled data is split into two parts: the *training set* and the *validation set*. During each epoch of training, the training set will be used to adjust the trainable parameters of the network. The validation set will be used to see if the network is not *overfitting*. An overfitted neural network has a very high performance on the training set but a low performance on the test set. This happens because the network is not able to generalize concepts from the training set and apply these concepts on the validation sets.

Early stopping

One way to prevent overfitting is by implementing *early stopping*. Early stopping finishes the training of the network as soon as the performance on the validation set does not increase anymore. The amount of training iterations the performance on the validation set is allowed to decrease before stopping training is called the *patience*. After the early stopping has run out of patience, it will revert back to the iteration where the performance on the validation set was the highest (see figure 2.3).

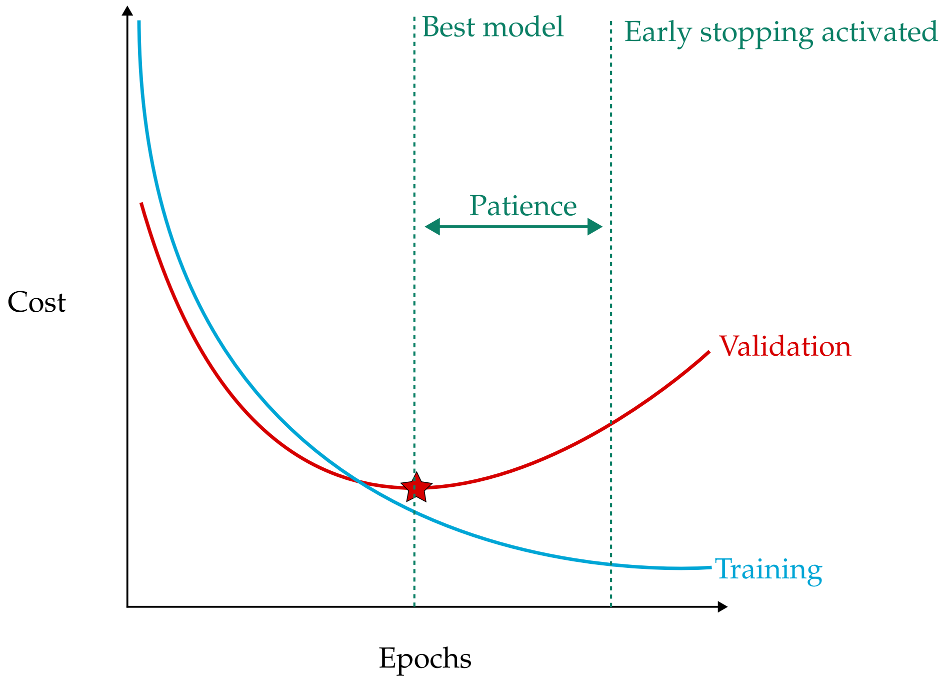

Figure 2.3: Early stopping activates when the cost of the validation set increases.

Batch normalization

Batch normalization is a technique commonly used in deep neural networks to improve training efficiency and generalization performance. It also regularizes the network reducing the need for other regularization techniques such as dropout [19]. It involves normalizing the activations of previous layers by adjusting and scaling them to have zero mean and unit variance. This normalization is applied to mini-batches of training examples during the training process.

One mini-batch B of size m has activation vectors $\mathbf{x_1}, ..., \mathbf{x_m}$. Let

$$\mathbf{x_i} = \begin{bmatrix} x_i^{(1)} \\ \vdots \\ x_i^{(d)} \end{bmatrix} \tag{2.15}$$

be a vector of d activations for sample i of mini-batch B. The batch normalization transform $BN_{\gamma,\beta}(x)$ is computed for each of the scalar activations $x_i^{(k)}$ of $\mathbf{x_i}$. The mini-batch mean $\mu_B^{(k)}$ and mini-batch variance $v_B^{(k)}$ are defined as:

$$\mu_B^{(k)} = \frac{1}{m} \sum_{i=0}^{m} x_i^{(k)}, \tag{2.16}$$

$$v_B^{(k)} = \frac{1}{m} \sum_{i=0}^{m} (x_i^{(k)} - \mu_B^{(k)})^2, \tag{2.17}$$

$\mu_B^{(k)}$ and $v_B^{(k)}$ are used to calculate the normalized activation $\hat{x}_i^{(k)}$ via

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{v_b^{(k)} + \varepsilon}}, \tag{2.18}$$

where ε is a small positive constant to prevent division by 0. In order to scale and shift the normalized activation a linear transform is applied:

$$y_i^{(k)} = \gamma^{(k)} \hat{x}_i^{(k)} + \beta^{(k)}, \tag{2.19}$$

where $\gamma^{(k)}$ and $\beta^{(k)}$ are learnable parameters. The batch normalization transform can now be written as:

$$BN_{\gamma,\beta}(x_i) = \begin{bmatrix} y_i^{(1)} \\ \vdots \\ y_d^{(d)} \end{bmatrix}. \tag{2.20}$$

It is still unclear how batch normalization helps the training of artificial neural networks. At first it was hypothesized by Ioffe and Szegedy that batch normalization reduces *internal covariate shift* [19]. Internal covariate shift refers to the change in the distribution of network activations that occurs during the training of neural networks. Due to changes in the parameters of earlier layers, the distribution of inputs to subsequent layers also changes. This is problematic because a neural network is essentially learning a mapping from the input distribution to the target distribution. By changing the distribution of earlier layers, the network constantly has to adapt which makes training more difficult. This can be more troublesome for deep neural networks since small changes in earlier layers get amplified as they propagate. Santurkar et al. have argued that batch normalization smoothens the hyperparameter space, allowing for more efficient traversal using an optimizer [20].

2.1.5. Convolutional Neural Networks

Images can also be used as an input for ANNs. Pixel intensities are represented as numbers in a matrix. Each number serves as an input to one neuron in the network. Since weights and biases are needed for every neuron in each layer, this is computationally very expensive. Therefore, it is more common to use a *convolutional neural network* (CNN) when working with images. As the name suggests CNNs rely on convolutions. In image analysis convolutions produce an output image from an input image using a kernel. The kernel is a matrix with weights that slides over the input image. Overlapping parts of the kernel and the input image are multiplied elementwise. That value is assigned to a new image at the location of the kernel. Dealing with boundaries of images can be done with padding. The parts of the kernel where there is no overlap with the image are taken to be zero. One kernel produces one image or feature map. In practice convolutional layers use multiple kernels to produce multiple feature maps. These feature maps serve as input to the next convolutional layer.

In a traditional ANN each neuron has a connection to each input pixel. In a CNN each neuron only has a connection to each input pixel within the kernel. Since the size of the kernel is way smaller than the image size, fewer computations have to be done. Furthermore, the kernel weights are shared between the neurons. This means less parameters have to be stored as well. This massive reduction in trainable parameters allows for more layers to be used in the network.

Convolutional layers typically contain a *pooling layer*. The pooling layer reduces the spatial dimensions of the image while retaining the most important features. This is done by dividing the input image in non overlapping square boxes. In *max pooling* [21] the maximum of each of these boxes is the output pixel at the location of that box. This reduction in spatial dimensions not only helps with the computational load, but it also makes the network somewhat translation invariant. If one pixel within the box has a high value, all other pixels in that box will not have an influence on the output of the max pooling layer.

If an image has gone through multiple convolutional layers with max pooling, the spatial dimensions are a lot smaller than the input image. If the output of the network should be a pixel-wise classification, the spatial dimensions of the output should be the same as the input image. Therefore, layers with *transposed convolutions* or *deconvolutions* are added after the final convolutional layer. Transposed convolutions consist of two steps: upsampling and convolution. During the upsampling step zeros are added in between each pixel to increase the image size. Then, a regular convolution is applied to this upsampled image. The stride of this convolution is 1, so the output of the layer has the same spatial dimensions as the upsampled image. These convolutions also have trainable parameters and are thus learned during training. Networks that only contain convolutions and no fully connected layers are called *fully convolutional networks*.

2.2. Models

2.2.1. U-net

U-net is a very popular network architecture used in image segmentation, due to its ability to combine fine details and high-level semantic information. The U-Net architecture derives its name from its U-shaped structure, which consists of a contraction and an expansion path. The contraction path reduces the spatial dimensions of the input image while extracting features via convolutions and max-pooling layers. In the expansion path a series of transposed convolutions and upsampling layers increase the spatial dimensions of the output of the contraction path. Furthermore, concatenations between features from the contraction and expansion path are incorporated in multiple layers. This facilitates the integration of lower level, highly detailed features with higher level, more abstract features [22].

The activation function used for the networks in this thesis is the LeakyRelu [23]. It is defined as follows:

$$LeakyRelu(x) = \begin{cases} x, & \text{if } x \geq 0, \\ \alpha \cdot x, & \text{otherwise,} \end{cases} \tag{2.21}$$

where α is 0.01 for all experiments.

The amount of background pixels is larger than the amount of pixels belonging to microtubules, which makes learning more difficult for the network. In order to account for this class-imbalance and to force the network to learn separation between touching objects, a weight map is introduced. This weight map is computed as:

$$w(x) = w_c(x) + w_0 \cdot \exp -\frac{(d_1(x) + d_2(x))^2}{2\sigma^2}, \tag{2.22}$$

with $w_c : \Omega \to \mathbb{R}$ is the weight map that accounts for the class-imbalance. $d_1 : \Omega \to \mathbb{R}$ is the distance to the border of the nearest object and $d_2 : \Omega \to \mathbb{R}$ the distance to the border of the second nearest object. The second term in equation 2.22 will be large if an object is near other objects. This increases the importance of microtubules in close proximity.

The U-net architecture will serve as a backbone to all the other networks that are used in this thesis. Furthermore, the U-net architecture as shown in figure 2.4 will be used as a comparison to the ISOO$_{\text{DL}}$ network described in section 2.2.2.

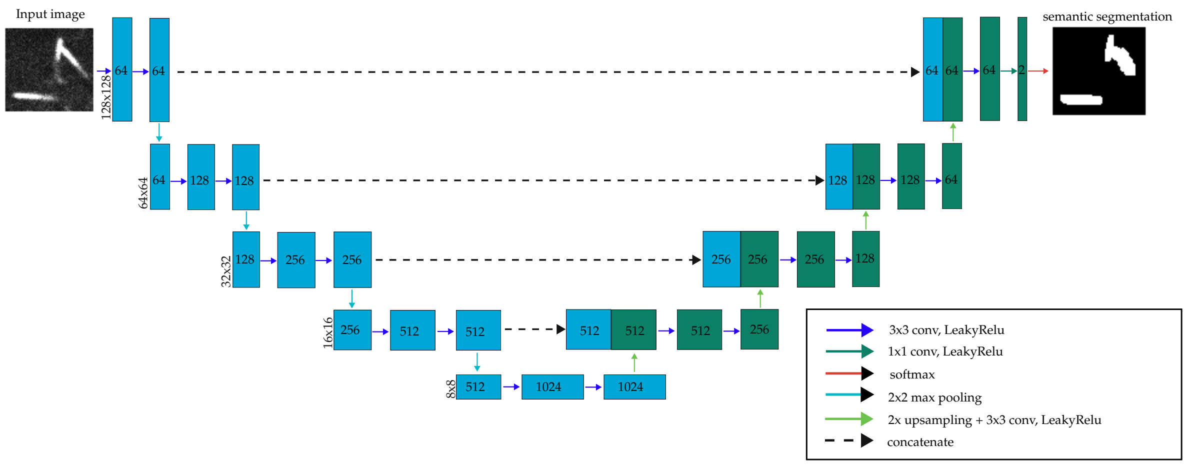

Figure 2.4: U-net structure used in this project. The numbers in the colored blocks correspond to the amount of feature maps at that position. The numbers on the side of the blocks correspond to the width and height of the feature maps at that position

Training

This section provides an overview of the paramater settings used during training of the U-net network. 90 synthetically generated images with a signal to noise ratio of 5 were used to train the network. These images were rotated and flipped to create 360 images in total. 250 of these images were used for training, and 90 for the validation during training. Please refer to table 2.1 for the other parameters used.

Table 2.1: Parameter Settings for the U-net network

Parameter	**Value**
Epochs	500 (early stopping reverted back to epoch 24)
Adam β_1	0.9
Adam β_2	0.999
learning rate	0.0001
Early Stopping Patience	10

2.2.2. ISOO$_{\text{DL}}$

Conventional convolutional neural networks, like U-net, have been shown to be a drastic improvement to pixel-wise semantic segmentation [24]. However, these networks are not capable of instance detection, and rely on the classification of pixels. This means that if two objects are overlapping, adjacent pixels from two different objects will have the same class. These objects will be merged in the segmentation mask.

In an attempt to improve the separation of objects in the segmentation mask the *ISOO$_{DL}$* (Instance Segmentation of Overlapping biological Objects) network structure was adapted [1].

The proposed ISOO$_{\text{DL}}$ network structure simultaneously solves two tasks of our problem: object detection and instance segmentation. It is based on the U-Net architecture with two independent expansion paths to solve detection and segmentation.

We define one object, in our case a microtubule, as $O_k := (m_k, \mathbf{c}_k, \mathbf{b}_k, \mathbf{p}_k, y_k)$ where $m_k : \Omega \rightarrow \{0, 1\}$ is the binary mask of the object with $\Omega \subset \mathbb{N}^2$. $\mathbf{c}_k \in \mathbb{R}^2$ is the center of the

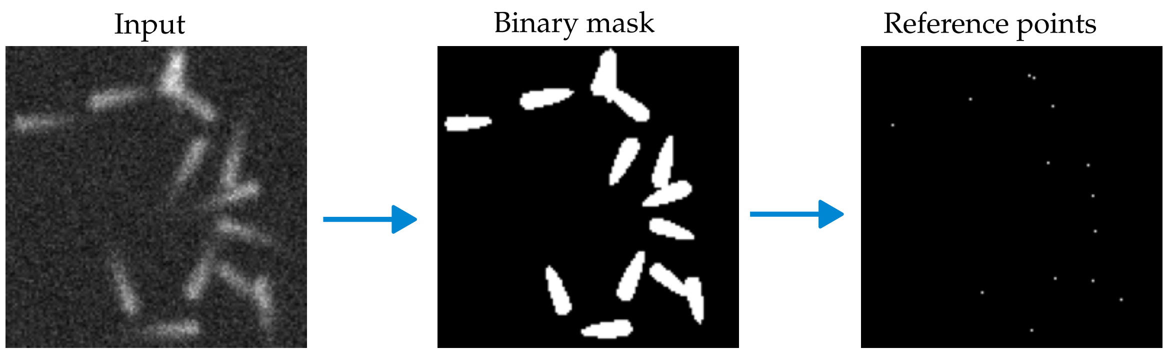

Figure 2.5: Object reference points are the centers of mass of the binary object masks

bounding box encompassing the object, $\mathbf{b}_k = (w_k, h_k)^\top \in \mathbb{R}^2$ are the width and height of the bounding box, $\mathbf{p}_k \in \mathbb{R}^2$ is the *reference point* of the object. Since some objects in the image are overlapping, their binary masks will be connected. Therefore, the location of the reference point of a microtubule is defined as the center of mass of that microtubule (figure 2.5). However, due to class imbalance the prediction of small reference points can become quite challenging. Therefore a dynamic reference point size was used, as proposed by Böhm, Tatarchenko, and Falk in [25]. Instead of fixing the size of reference points, we allow them to shrink if they are in close proximity to another reference point. Instead of a single pixel, we can now use disks with a maximum size of r_{max} to show where reference points are. To do this we compute the euclidean distance. To determine how large a disk should be, we compute the euclidean distance to the nearest reference point of another object

$$d_k = \min(\frac{d_k}{d} - 2, r_{max}). \tag{2.23}$$

Furthermore, we also introduce per pixel loss weights based on the size of the disks, to account for the impact of varying disk size on the loss:

$$\omega_k := \min(\pi(r_{max}^2 - \hat{r}_k^2), 10). \tag{2.24}$$

Additionally, weights as described in 2.22 are added. The value of d_2 was set to 0 because it decreased performance of the network. ω_0 was set to 0.3 and σ was set to 25.

$y_k \in \{1, \dots, C\}$ is the class label of the pixel. The set of all objects is $O = \{O_1, \dots, O_k\}$. Predicted variables are indicated with a hat, and primed variables indicate 3D representations. $\mathbf{x} = (x, y)^\top \in \Omega$ are the 2D coordinates of the image. $\mathbf{x}' = (\mathbf{x}^\top, x)^\top \in \Omega \times \mathbf{Z}$ are the corresponding 3D coordinates.

Object detection

The object detection path has two outputs: object location and object size. The network is trained to classify each pixel as either background or reference point. Furthermore, the network also predicts the size of a bounding box that encompasses an object. If the network is given an input image, the object location output will produce a score map of real positive values. These values represent the likelihood of a pixel belonging to either the background or an object reference point. Since we need a binary classification of pixels, a threshold is applied. The centers of mass of the connected components in this binarized output are used as object centers. The prediction of bounding box width and height was unsuccessful in our experiments, and therefore the size was fixed to 20x20 pixels. This bounding box approximation encompasses microtubules fully, which is important to successfully extract all information out of the predicted sheared object masks described in section 2.2.2.2.

Object segmentation

In the case of overlapping objects we are dealing with multiple labels per location. In order to teach the model that these are separate objects we lift the domain of the labels from 2D to 3D. This lifting is done via

$$m'_k(x') = \begin{cases} m_k(x) & z = 0, \\ 0 & \text{otherwise.} \end{cases}$$

Then a shearing is applied to all coordinates within a bounding box corresponding to an object. The sheared bounding box centers $c'_k = (c_k^\top, 0)^\top$ remain at $z = 0$. All masks of the separate objects are described in one function $m'^x : \Omega \times \mathbb{Z} \to (0,1)$ where

$$m'^x(x') := \max_{k \in 1,\dots,K} (m'_k \circ (T_k^x)^{-1})(x'),$$

$T_k^x : \mathbb{R}^3 \to \mathbb{R}^3$ is the object specific shear transformation. Superscripts $.^x$ and $.^y$ indicate shearing in xz direction or yz direction respectively. The shear transformation T_k^x can be described as follows:

$$T_k^x = (T_{c_k})^{-1} \circ T_s^x \circ T_{c_k}.$$

T_{c_k} shifts the object bounding box center c_k to the origin. T_s^x shears the pixels inside the bounding box with a fixed angle of 45 degrees. In order to prevent the size of the transformed data to become prohibitively large a maximum size is set at 23 slices in the z dimension. Should a bounding box have a size that is larger than 23 pixels, the outer pixels will be lifted to the highest or lowest z-slice depending on the shearing direction. T_k^y is constructed similarly but with T_s^y shearing in yz direction.

Lifting the objects in xz-direction solves the problem of separation in the y direction. In order to separate the objects in x direction they are also lifted in yz-direction. Combining these two outputs will lead to full separation in 3D space. The last output of the network is a semantic segmentation. This segmentation is not used in the post-processing steps but it stabilizes the training.

All outputs of the network were trained using a weighted binary cross entropy (BCE) as a loss function. The weighted BCE is defined as follows:

$$BCE(y, \hat{y}) = \omega \cdot (-y \log(\hat{y}) - (1-y)\log(1-\hat{y})), \tag{2.25}$$

where $\hat{y}$ is the predicted probability of output y. For the reference point predictions, ω is as described in section 2.2.2. For the segmentation outputs ω increases the weights of the positive class, such that the contribution of the positive class on the loss is equal to the contribution of the negative class.

For an overview of the object segmentation path see figure 2.6

Post-processing

The object detection path of the network predicts the location of objects in the input image. For each of these predicted objects the shearing transformation has to be undone by using the object specific T_k^x and T_k^y. This maps the probabilities back to the $z = 0$ plane:

$$p_k^x(x') = (\hat{m}'^x \circ T_k^x)(x'),$$

and

$$p_k^y(x') = (\hat{m}'^y \circ T_k^y)(x').$$

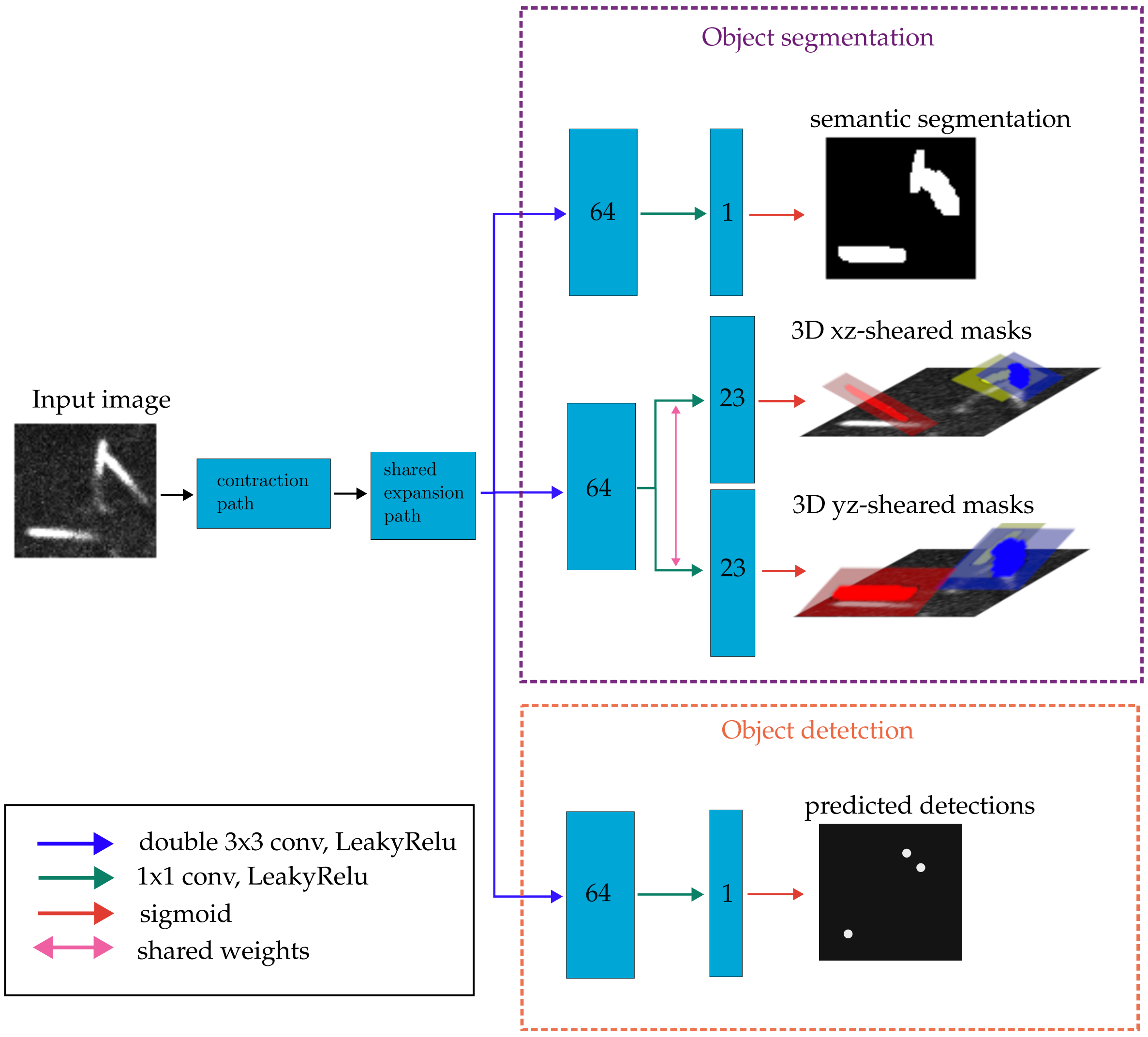

Figure 2.6: Expansion paths of the ISOO$_{\mathrm{DL}}$ network. The numbers in the rectangles correspond to the amount of channels in that point of the network.

In order to obtain 2D probability maps the average joint probability $\frac{p_k^x(x')+p_k^y(y')}{2}$ is projected along the z-axis weighted by a Gaussian $N_{0,\sigma}$, with σ equal to 1.

The gaussian is applied to account for any localization inaccuracies. Then Otsu thresholding [26] is applied per predicted map to get the final binary masks.

Overlap

We also added an extra output for the ISOO$_{\mathrm{DL}}$ network that predicts the amount of overlap that is present at each pixel. If a pixel belongs to two microtubule segmentations, then the overlap at that pixel is also 2. The overlap output of the network has $N + 1$ channels, where N is the maximum amount of overlap present in the data. For this project, N was set to 4. The loss function used on this output was also a binary cross-entropy. The final predicted amount of overlap is determined by taking the index of the highest predicted class across the channel dimension.

Training

This section provides an overview of the parameter settings used during training of the ISOO$_{\text{DL}}$ network. 90 synthetically generated images with a signal to noise ratio of 5 were used to train the network. 81 of these images were used for training, and 9 for the validation during training. We refer to table 2.2 for the other parameters used.

Table 2.2: Parameter settings for the ISOO$_{\text{DL}}$ network

Parameter	**Value**
r_{max}	3
Epochs	500 (early stopping reverted back to epoch 101)
Adam β_1	0.9
Adam β_2	0.999
learning rate	0.00001
Early Stopping Patience	10

2.2.3. Overlap network

Since the ISOO$_{\text{DL}}$ network is able to predict where microtubules are overlapping, another network trained on overlapping microtubules is proposed. This network can be used to further improve segmentation of overlapping microtubules. Unfortunately there is very little annotated data of exclusively overlapping microtubules. Therefore, a synthetic data generator was used to generate data with overlapping microtubules. Since the lifting and shearing of microtubules in the ISOO$_{\text{DL}}$ network did not produce satisfactory results, a different separation was used. The segmentation map of each individual microtubule is put in a different layer.

This does mean that the network can only predict segmentation maps of N different microtubules, as the output has N different layers. If there are less than N microtubules in the image, some layers will contain no segmentation in the form of a black image.

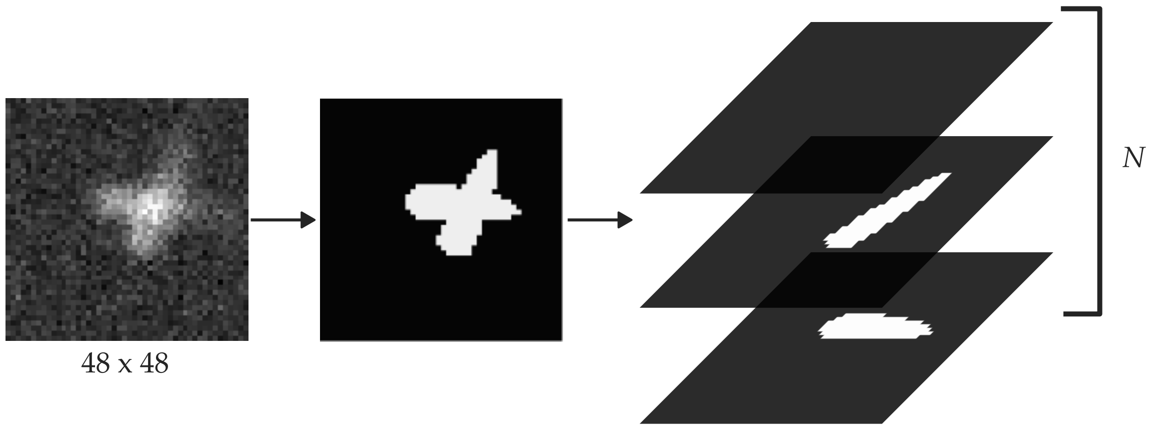

Figure 2.7: The individual microtubule segmentation masks are put in different channels.

Due to the fact that the size of the images is smaller than in the ISOO$_{\text{DL}}$ network, the network structure has to be adapted. The max pooling operations used in the contraction path reduce the spatial dimensions of the image. If no modifications would have been made, the image would be one pixel at the deepest part of the network. The upsampling and deconvolutions would not be able to extract much relevant information out of this pixel. The input images have a size of 48x48 pixels, and at the end of the contraction path the size is 12x12 pixels. The input of this network is the amount of overlap predicted by the ISOO$_{\mathrm{DL}}$ network concatenated with the raw input image. The network also predicts a regular segmentation with a BCE as loss function, which aids in training. In order to further improve the performance of this network and to account for the change in data structure, some additional loss functions were added.

Loss functions

The primary loss function used for the segmentation output, was a cross-entropy loss. The activation function of this prediction is a softmax, which ensures that the probabilities of belonging to each class sum up to 1:

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}, \tag{2.26}$$

where j is the amount of classes.

The cross-entropy loss (CEL) is defined as:

$$CEL = -\sum_{i=1}^{N} y_i \cdot \log \hat{y}_i, \tag{2.27}$$

where N is the amount of pixels, y_i and $\hat{y}_i$ are the ground truth and predicted probabilities respectively. Furthermore we also added a weight map, which consisted of weighing all pixels that belong to microtubules other than the one of the current class.

The *dice loss* was used as an additional loss function for the segmentation of separated microtubules. It originates from the Sørensen-Dice coefficient [27, 28] and was introduced to computer vision by Milletari, Navab, and Ahmadi [29]. The Sørensen-Dice coefficient is defined as:

$$D = \frac{2\sum_i^N p_i g_i}{\sum_i^N p_i^2 + \sum_i^N g_i^2}, \tag{2.28}$$

where p_i is the predicted binary mask and g_i is the ground truth. Cross-entropy loss is calculated per pixel without knowing the values of adjacent pixels. Therefore, cross-entropy loss does not consider global information of the image, which dice loss is able to do.

We also tried applying a *sensitivity-specificity loss* (SLL), which uses the sensitivity and specificity of the segmentation predictions.

$$SSL = \omega \cdot sensitivity + (1 - \omega) \cdot specificity, \tag{2.29}$$

where the sensitivity and specificity are defined as:

$$sensitivity = \frac{TP}{TP + FN}, \tag{2.30}$$

$$specificity = \frac{TN}{TN + FP}. \tag{2.31}$$

The variable ω was set to 0.3.

Additionally, we also use *focal tversky loss* (FTL). This loss function down-weights easy samples and thus focuses more on hard cases using the γ coefficient.

$$FTL = \sum_c (1 - TI_c)^{\gamma}, \tag{2.32}$$

where TI is the tversky index:

$$TI(\beta) = \frac{TP}{TP + \beta FN + (1-\beta)FP}, \tag{2.33}$$

where TP, FN and FP are defined in section 2.3.1. For this project, β was set to 0.7 and γ was set to 2.

Furthermore, we also want to ensure that the predicted microtubule segmentations sum up to the amount of overlap. Therefore, we compute the *mean squared error* between the sum of the predicted microtubule segmentations and the amount of overlap per pixel. The mean squared error is defined as:

$$MSE = \frac{1}{N}\sum_{i}^{N}(p_i - g_i)^2. \tag{2.34}$$

On top of that we also applied a binary cross entropy to the maximum projection of the N microtubule segmentations. This ensures that there are no pixels that should belong to a microtubule that are not predicted by the network.

The network also predicts the overlap that was used as one of the input images. The activation function of this prediction is a softmax, and the loss a regular cross-entropy loss. The loss used for this output was a regular cross-entropy loss.

Post-processing

The final segmentation output is produced by applying a softmax to the predicted segmentations. Due to the fact that all probabilities now add up to 1, we have to do some post-processing to assign pixels that should belong to more than 1 microtubule. First we assign pixels to the class of highest probability. However, if the predicted probability is relatively low, for example 0.4, that means that there is some uncertainty as to which class a pixel belongs to. That is why we also assign pixels to the classes where the predicted probability is between 0.2 and 0.5. These values were picked, because in the data of this project the maximum amount of microtubules N was set to 3. If a microtubule belongs to 2 classes, the highest possible non-maximum probability is 0.5. If a microtubule belongs to 3 classes, all probabilities will be somewhere around 0.33. In order to account for some variation between probabilities we have set the lower bound to 0.2.

2.3. Performance evaluation

Various metrics are utilized to asses the performance of the ISOO$_{\text{DL}}$ network. The network produces both locations of microtubules and segmentation of microtubules, and both will be evaluated separately.

2.3.1. Evaluation segmentation

The quantification of the performance on the segmentation task can be done using a *receiver operating charactersistic (ROC)* curve. A ROC curve requires the computation of true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN). In the case of semantic segmentation they are defined as follows:

1. True positives (TP): The number of pixels that are correctly predicted as positive.
2. False positives (FP): The number of pixels that are incorrectly predicted as positive, while they belong to the negative class.
3. True negatives (TN): The number of pixels that are correctly predicted as negative.

4. False negatives (FN): The number of instances that are incorrectly predicted as negative, while they belong to the positive class.

In our case the positive class is a pixel belonging to a microtubule and the negative class is a background pixel. Using these numbers we can calculate the *true positive rate (TPR)* and *false positive rate (FPR)*:

$$TPR = \frac{TP}{TP + FN}, \tag{2.35}$$

$$FPR = \frac{FP}{FP + TN}. \tag{2.36}$$

In the ideal situation, a model has $TPR = 1$ and $FPR = 0$ figure (2.8). The different points on the ROC curve are obtained by varying the parameters of the model, in our case the threshold that is applied to the network outputs. This threshold changes the classification of pixels and thus the TPR and FPR.

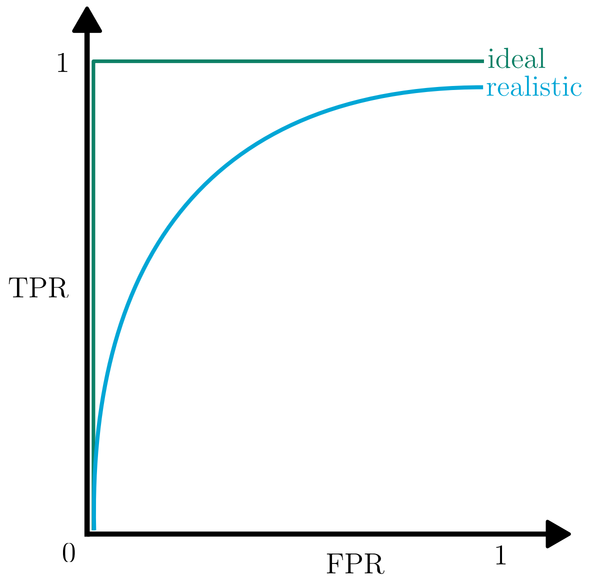

Figure 2.8: A perfect and more typical ROC curve, the perfect ROC curve has $TPR = 1$ and $FPR = 1$.

In order to quantify the detection performance of microtubules we also want to compute the TP, FP, TN and FN. However, this is not a straightforward task for the locations of microtubules. Therefore, we first discuss how predictions of microtubules are assigned to ground truth locations.

2.3.2. Assigning locations

The network does not perform a pixelwise classification for the detection of microtubules, but produces a set of coordinates. These coordinates will not be at the exact location of the ground truth coordinates, and therefore the performance would seem very low. This is why the predicted coordinates have to be assigned to ground truth coordinates, given they are in close proximity to the ground truth coordinates. The problem at hand can be seen as a variant of the assignment problem, which can be effectively solved by applying the Kuhn-Munkres algorithm [30]. An adjacency matrix is computed for the combinations of detections with ground truths, with a weight corresponding to the distance between the predicted detection and ground truth. If the distance between detection and ground truth is larger than 4 pixels, the weight is set to infinity. One the adjacency matrix has been established, the Kuhn-Munkres algorithm will assign each predicted detection to a ground truth location while minimizing the sum of the weights of chosen assignments (figure 2.9).

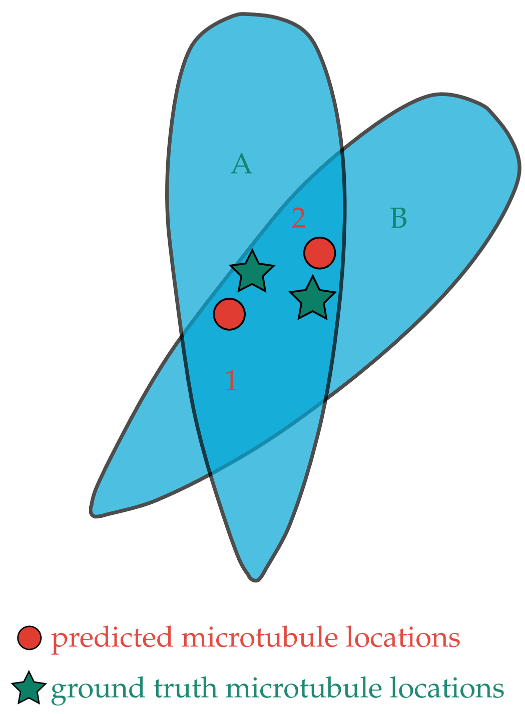
A
B
2
1
● predicted microtubule locations
★ ground truth microtubule locations

Figure 2.9: An example of non-trivial location assignment. Both predictions are within the boundaries of the microtubule. Therefore, the Kuhn-Munkres algorithm is used to assign the predicted locations to the ground truth locations.

Using this information we can compute the TP, FP, TN and FN:

1. True positive (TP): A true positive is defined as a detection made at the location of the microtubule tip.
2. False positive (FP): A false positive is a detection that is incorrectly predicted as positive. This means that the detection is made too far away from the actual location of the microtubule tip.
3. True negative (TN): Since the network only predicts locations of microtubule tips, and not of background pixels, there is no good way to define true negatives.
4. False negative (FN): A false negative means that a ground truth location was not assigned a predicted location by the Kuhn-Munkres algorithm.

2.3.3. Free-response Receiver Operator Characteristic

Since there is no way to define true negatives of detections, an adjustment has to be made to the ROC curve. Therefore, we use the *Free-response Receiver Operator Characteristic (FROC)*, which uses a modified false positive ratio $FPR*$:

$$FPR* = FP. \tag{2.37}$$

This FPR* is the mean number of FP locations per image. In contrast to the FPR used in a ROC curve, FPR* can exceed a value of 1.

2.3.4. Jaccard similarity coefficient

We are also going to evaluate the segmentation results based on the *Jaccard similarity coefficient* (JSC). This quantifies the overlap between ground truth segmentation and predicted segmentation:

$$JSC = \frac{TP}{TP + FP + FN}. \tag{2.38}$$

The JSC only focuses on the predicted and ground truth positive class, and is thus more robust to class imbalance. We will use the JSC to find the optimal segmentation threshold. Furthermore, we can also make a curve out of JSC values using different segmentation thresholds.

2.3.5. Detection localization

In order to compare the quality of TP detections, we are also going to asses the inaccuracy of detections. This will be done by calculating the mean squared error between the ground truth detections, and their assigned predicted detections.

2.3.6. Evaluation segmentation bounding boxes

One further adjustment has to be made in order to evaluate the models accurately. Since the ISOO$_{\mathrm{DL}}$ model computes segmentation maps for each bounding box, it makes sense to also look at segmentation of individual bounding boxes. This ensures that we quantify segmentation of only the instances. However, this only looks at detections that are made by the network and compares the segmentations resulting from these detections with the assigned ground truth segmentations. This segmentation is still a pixelwise classification and thus a normal ROC curve can be made in the same way as in 2.3.1.

Chapter 3 Results

In this chapter we showcase the outcomes of the experiments introduced in subsections 1.5.1 and 1.5.2. First we explain the reasoning behind experiment 1, as we have to evaluate many aspects of the ISOO$_{\text{DL}}$ network separately. Subsequently we present both the qualitative and quantitative results of our first experiment. We also present the results from experiment 2.

3.1. Experiment 1

In this section we explain the reasoning behind the results of experiment 1. In section 4.1 we interpret the presented results.

In order to compare the performance of microtubule detection, we made a FROC curve of the U-net network and the ISOO$_{\text{DL}}$ network. We made a curve of the MSE between ground truth detections and their assigned predicted detections at different detection thresholds (figure 3.6).

In figure 3.4 we can see the ROC curves for the segmentation of an entire image with microtubules. In the case of the U-net network this is a comparison between the segmentation prediction and ground truth segmentation. In the case of the ISOO$_{\text{DL}}$ network this is a comparison between the segmentation prediction *after* all post-processing steps have been applied and the ground truth segmentation. Figure 3.1 illustrates how the ROC curves for the segmentation of a full image of the ISOO$_{\text{DL}}$ network are made.

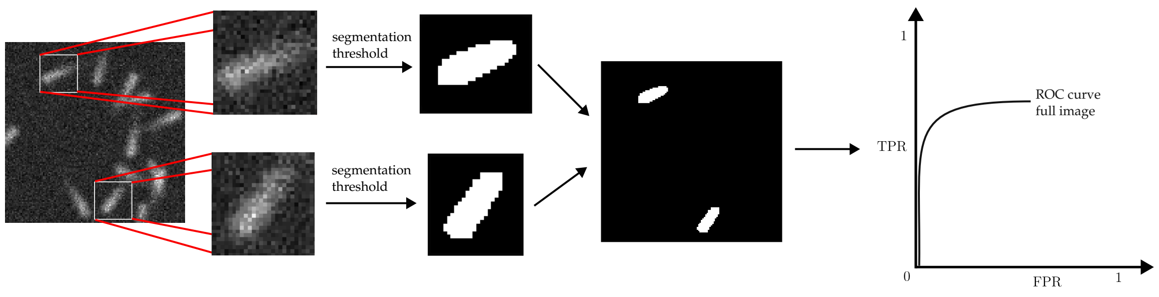

Figure 3.1: The ROC curves of the full image are made by comparing the segmentation of the entire image against the ground truth.

All pixels in the image are either negatively classified as background, or positively classified as microtubule. However, the ISOO$_{\text{DL}}$ network only segments the pixels in a bounding box surrounding a predicted detection location. This means that if we decrease the segmentation threshold we will only see more positively classified pixels in these boxes. Outside of these boxes, all pixels are negatively classified. If our amount of detected microtubule locations is unequal to the ground truth or in wrong locations, we can never reach a TPR and FPR of 1. An illustration of this is made in figure 3.2. We can look at this as an analog of the FROC curve, where the TPs are not reaching 1.

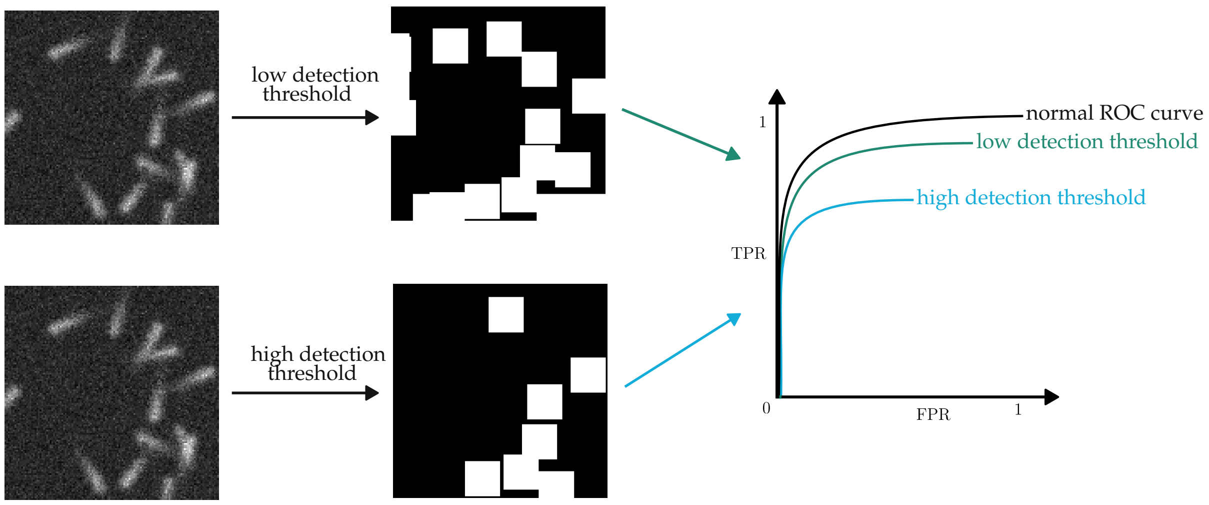

Figure 3.2: At low detection thresholds, the ISOO$_{DL}$ network produces many detections and thus more pixels are influenced by changing the segmentation threshold. At high detection thresholds, only few detections are made. At low segmentation thresholds, only the pixels within the boxes will be classified as positive. This means that we will never be able to positively classify pixels that are not inside of the bounding boxes. Therefore, the maximum achievable TPR and FPR at low detection thresholds are higher than at high detection thresholds. In order to get a normal ROC curve, that reaches a TPR of 1 and a FPR of 1, we would have to detect every microtubule, and also encapsulate every pixel of that microtubule inside the bounding box.

On top of that, the ISOO$_{DL}$ network is able to assign pixels where microtubules are overlapping to two different instances. This is not taken into account in the ROC curves of the full image.

Therefore, we are also going to look at ROC curves of the segmentation of individual bounding boxes. Now we are comparing the segmentation of individual microtubules, to the ground truth microtubule segmentation that was assigned by the Kuhn-Munkres algorithm. This does mean we are only able to compare TP microtubule detections. These curves *are* able to reach a TPR and FPR of 1, because the entire image is affected by changing the segmentation threshold. Figure 3.3 shows how the ROC curves of the detected bounding boxes are made.

Furthermore, we have also made multiple curves of the JSC at different segmentation thresholds. The points within one curve are JSC values of the predicted segmentation of individual bounding boxes compared to the ground truth segmentations. Different curves have different thresholds for microtubule detection predictions. This allows us to compare the shape of curves at different detection thresholds.

In order to determine the influence of the weighing in the post-processing steps, we will make ROC and JSC curves using a Gaussian with σ set to 0.5.

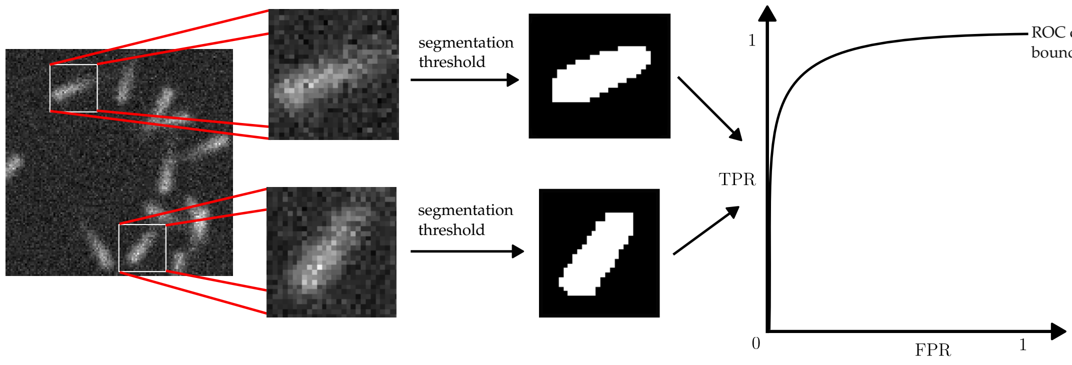

Figure 3.3: The ROC curves of only the detected boxes are made by comparing the detected boxes against their assigned ground truth boxes. The assignment in done using the Kuhn-Munkres algorithm.

3.1.1. Quantitative results experiment 1

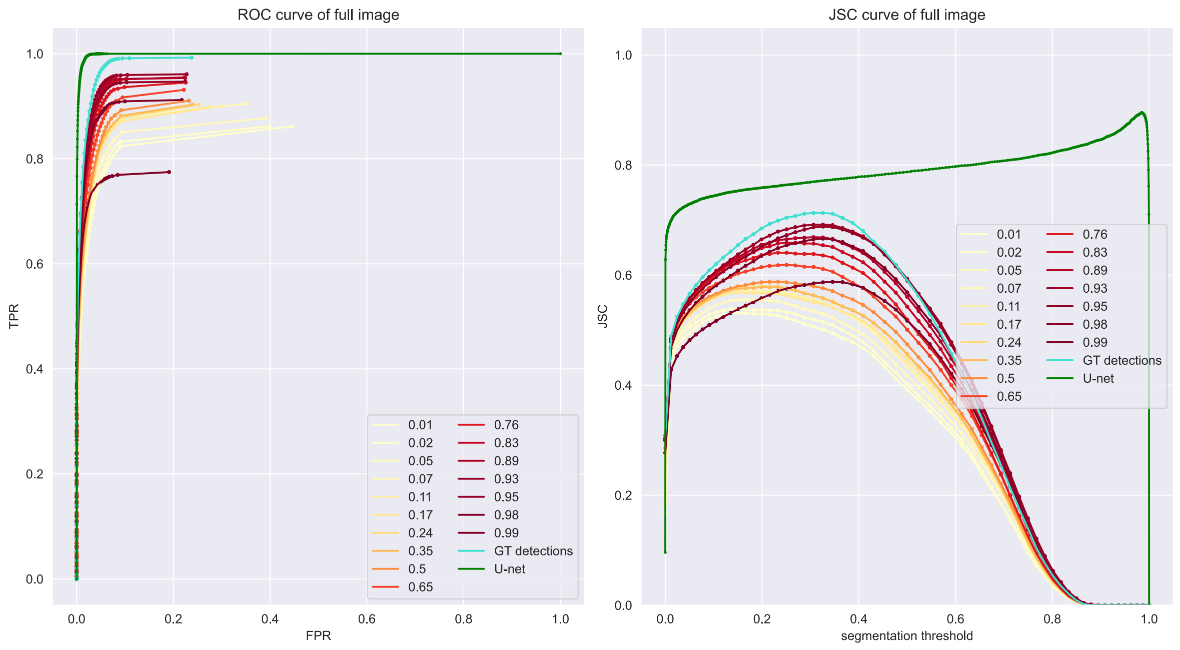

Figure 3.4: ROC and JSC curves of the ISOO$_{DL}$ and U-net network of the segmentation of 8 128x128 synthetic images. Furthermore, there is also a curve of the ground truth detections with the predicted segmentations.

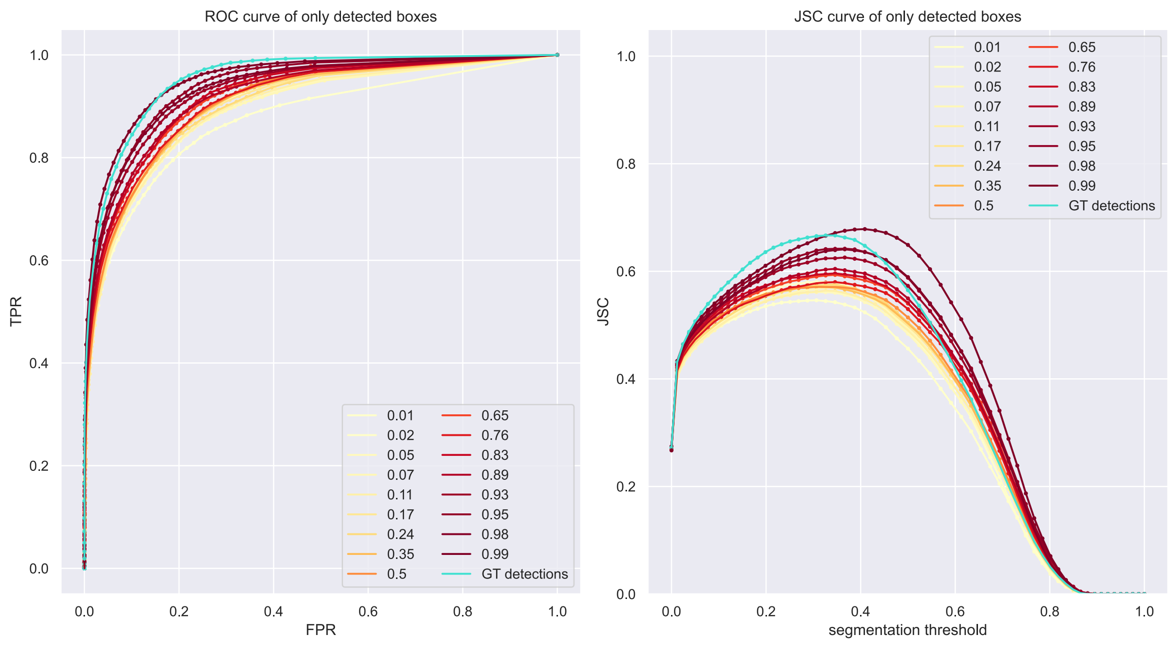

Figure 3.5: ROC and JSC curves of the $ISOO_{DL}$ and U-net network of the segmentation of only the bounding boxes of 8 128x128 synthetic images. Furthermore, there is also a curve of the ground truth detections with the predicted segmentations.

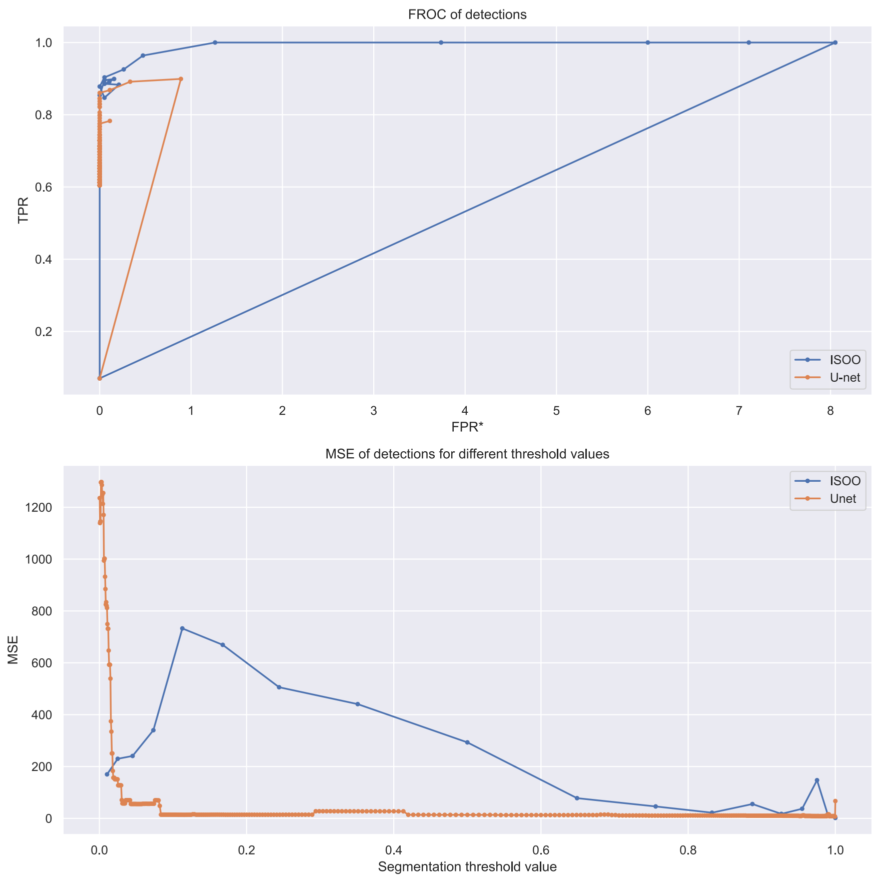

Figure 3.6: FROC and MSE curves of the $ISOO_{DL}$ and U-net network.

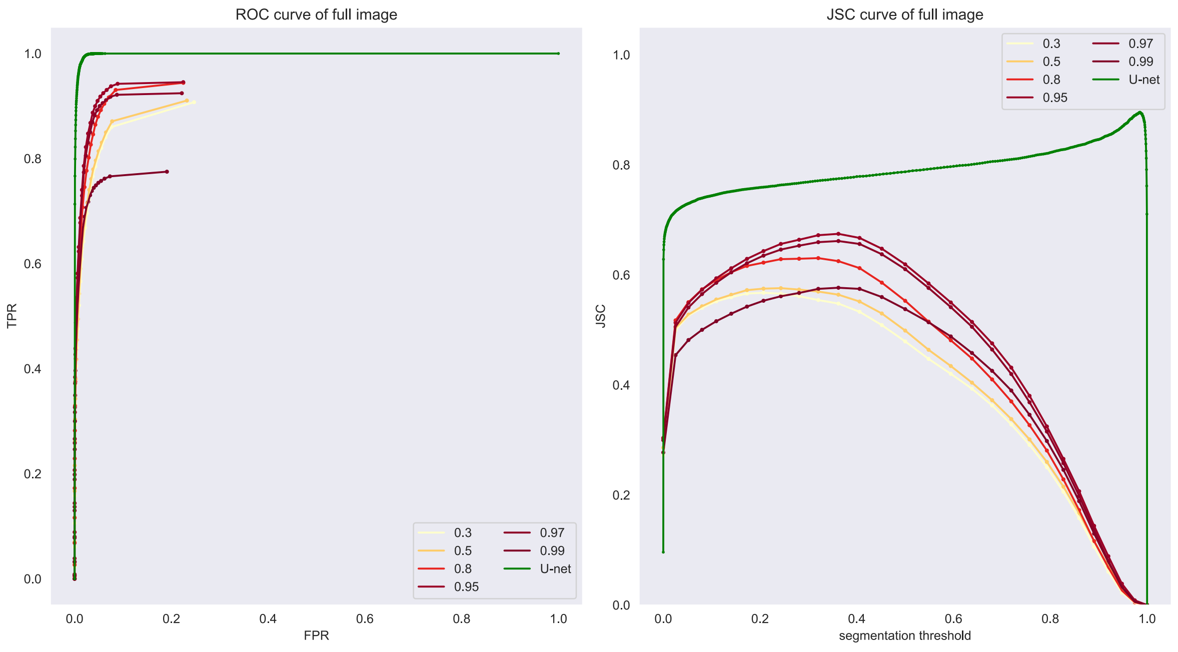

Figure 3.7: ROC and JSC curves of the ISOO$_{DL}$ and U-net network of the segmentation of 8 128x128 synthetic images. The σ of the Gaussian in the post-processing operations was set to 0.5.

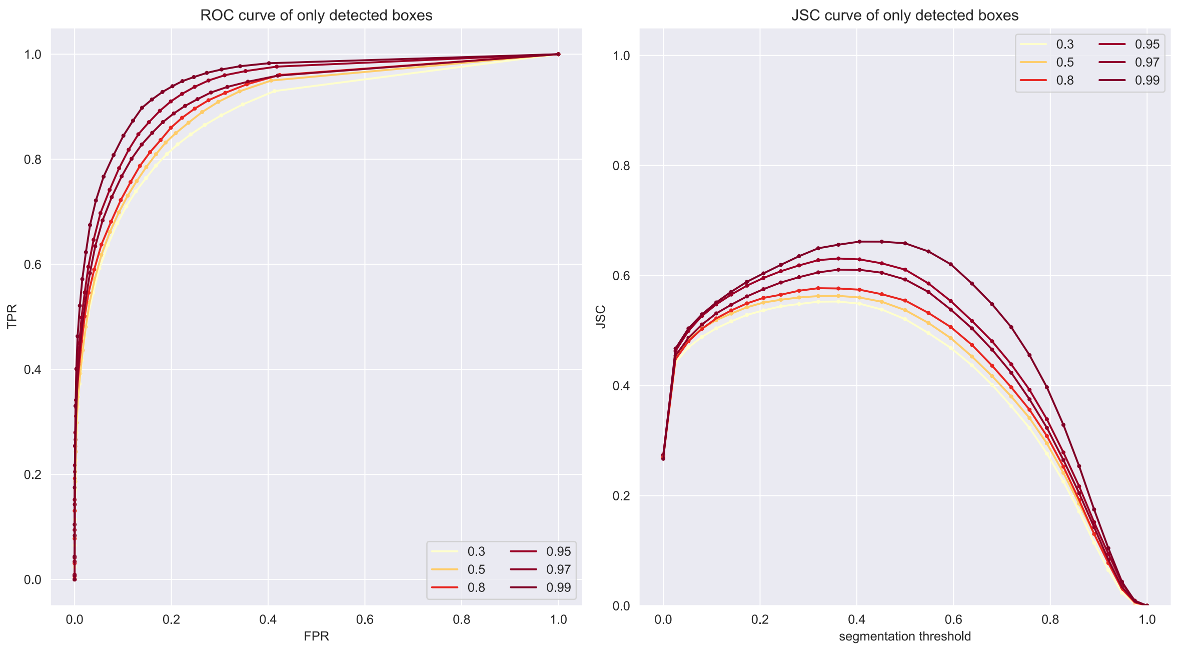

Figure 3.8: ROC and JSC curves of the ISOO$_{DL}$ and U-net network of the segmentation of only the bounding boxes of 8 128x128 synthetic images. The σ of the Gaussian in the post-processing operations was set to 0.5.

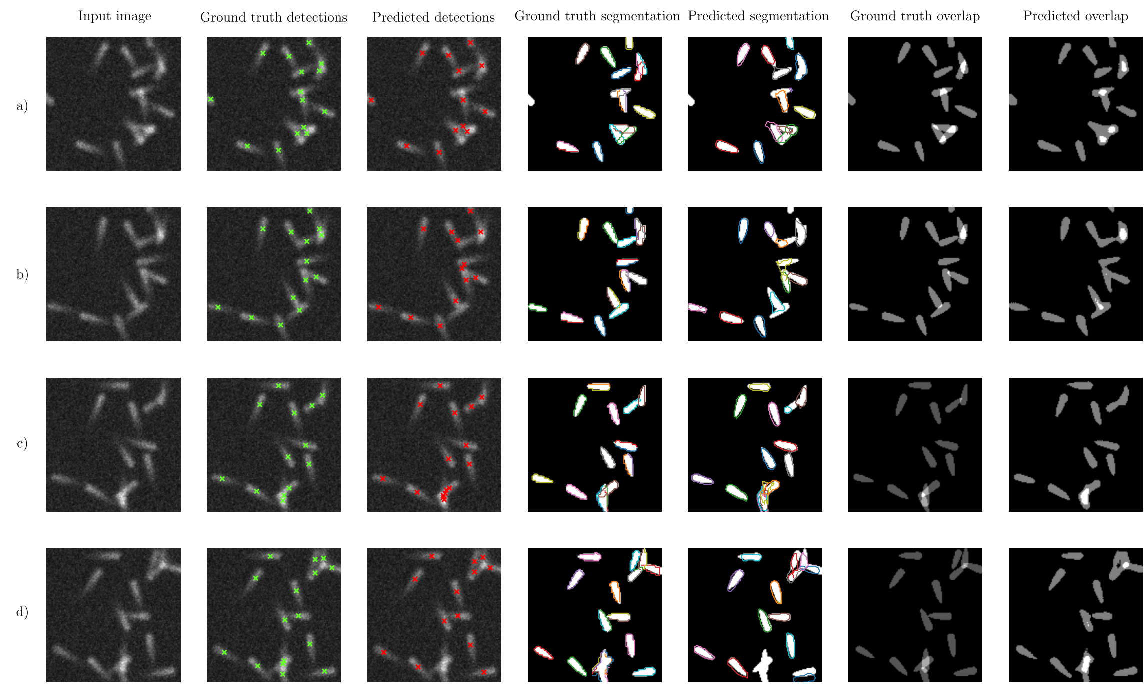

Figure 3.9: Qualitative results of the ISOO$_{DL}$ network, the detection threshold is 0.95 and the segmentation threshold is 0.4.

3.1.2. Qualitative results experiment 1

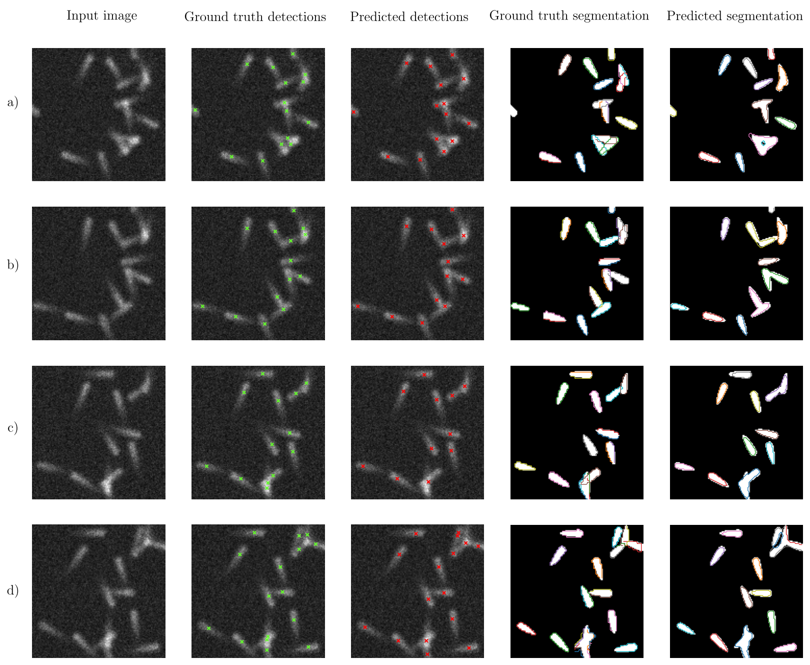

Figure 3.10: Qualitative results of the U-net network, the detection threshold is 0.9999 and the segmentation threshold is 0.995.

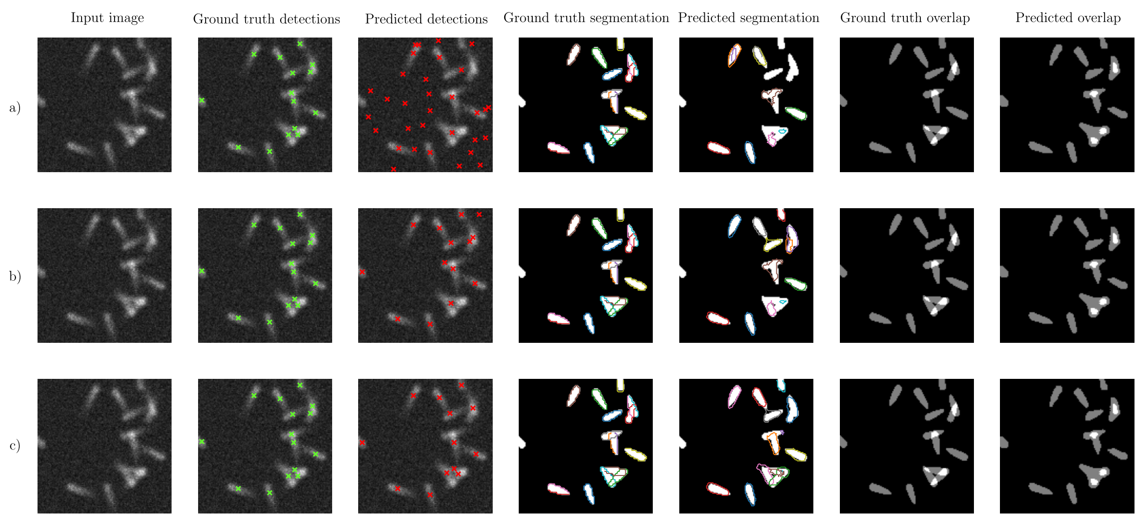

Figure 3.11: Qualitative results of the ISOO$_{DL}$ network. a: detection threshold is 0.001, b: detection threshold is 0.3, c: detection threshold is 0.95. The segmentation threshold is 0.4 for all images.

3.2. Experiment 2

In this section we present the results of experiment 2. Figures 3.12, 3.13, 3.14 and 3.15 show the segmentation results of the networks. Tables 3.1, 3.2, 3.3 and 3.4 list the weighing of the different loss functions. These networks were trained on 80 images, with 20 images used as validation. We also trained the best performing network on 800 images, with 200 images used as validation. The resulting segmentations are shown in figure 3.16.

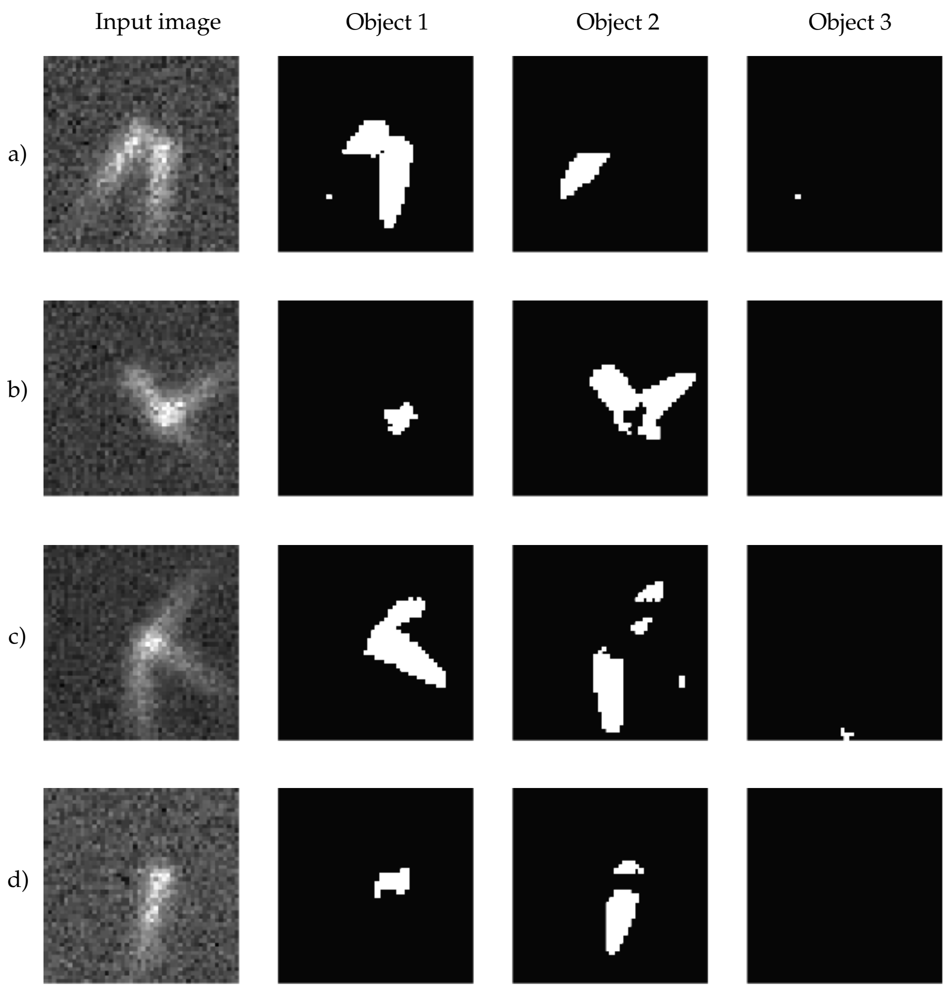

Figure 3.12: Qualitative results of the overlap network. The three objects represent the three classes that are predicted by the network. If the network would work perfectly, every microtubule segmentation would occupy a single image. The loss weights can be found in 3.1.

Table 3.1: Loss weights and learning rate for the overlap network in figure 3.12

Loss	**Value**
learning rate	0.001
Dice loss	100
Weighted CEL	1
MSE sum	100
CEL overlap	100
SensSpec	10
FTL	10
Maximum projection	100
BCE semantic	60

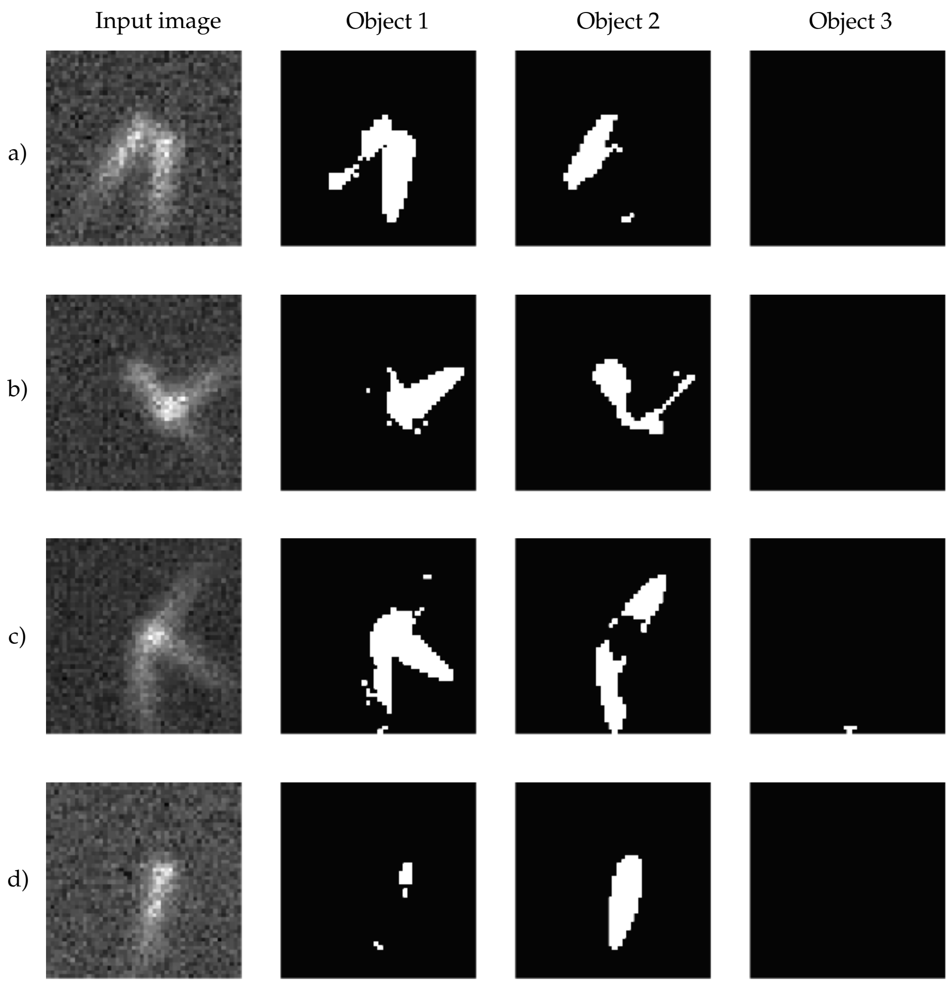

Figure 3.13: Qualitative results of the overlap network. The three objects represent the three classes that are predicted by the network. If the network would work perfectly, every microtubule segmentation would occupy a single image. The loss weights can be found in 3.2.

Table 3.2: Loss weights and learning rate for the overlap network in figure 3.13

Loss	**Value**
learning rate	0.001
Dice loss	1000
Weighted CEL	1
MSE sum	1000
CEL overlap	1
SensSpec	0
FTL	0
Maximum projection	0
BCE semantic	0

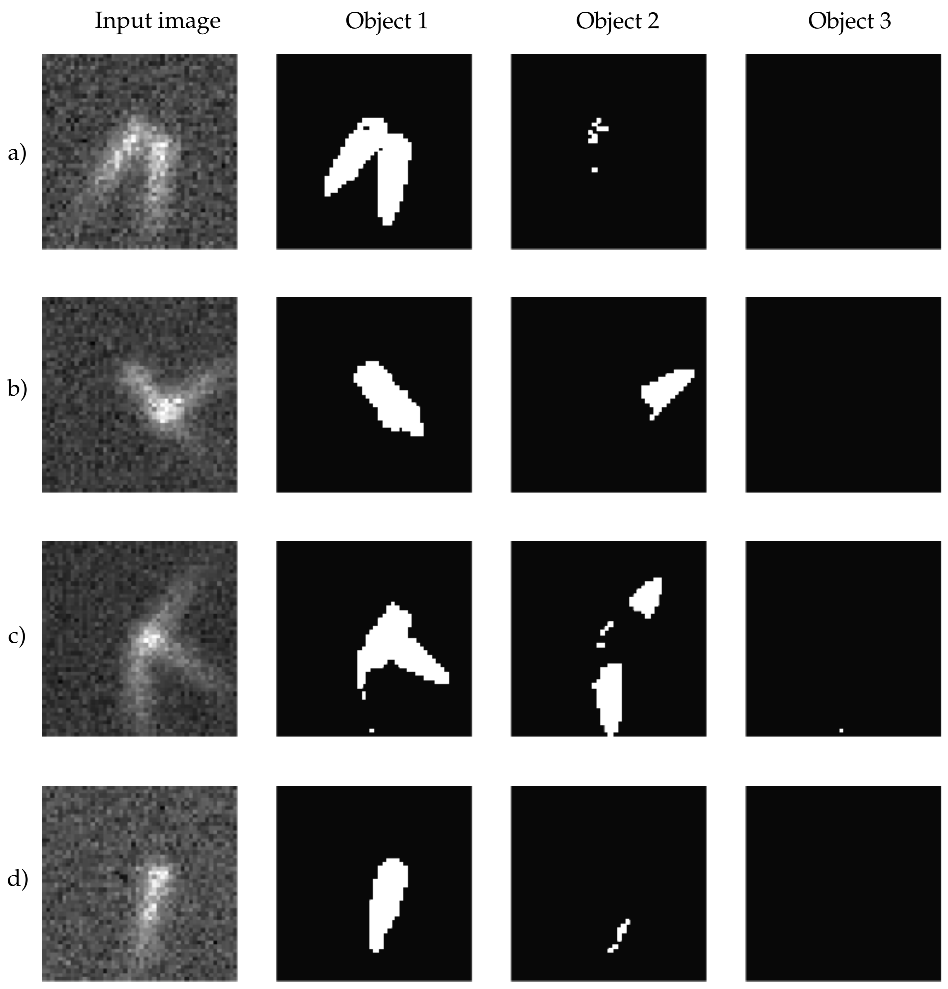

Figure 3.14: Qualitative results of the overlap network. The three objects represent the three classes that are predicted by the network. If the network would work perfectly, every microtubule segmentation would occupy a single image. The loss weights can be found in 3.3.

Table 3.3: Loss weights and learning rate for the overlap network in figure 3.14

Loss	**Value**
learning rate	0.001
Dice loss	0
Weighted CEL	1
MSE sum	100
CEL overlap	10
SensSpec	10
FTL	100
Maximum projection	20
BCE semantic	30

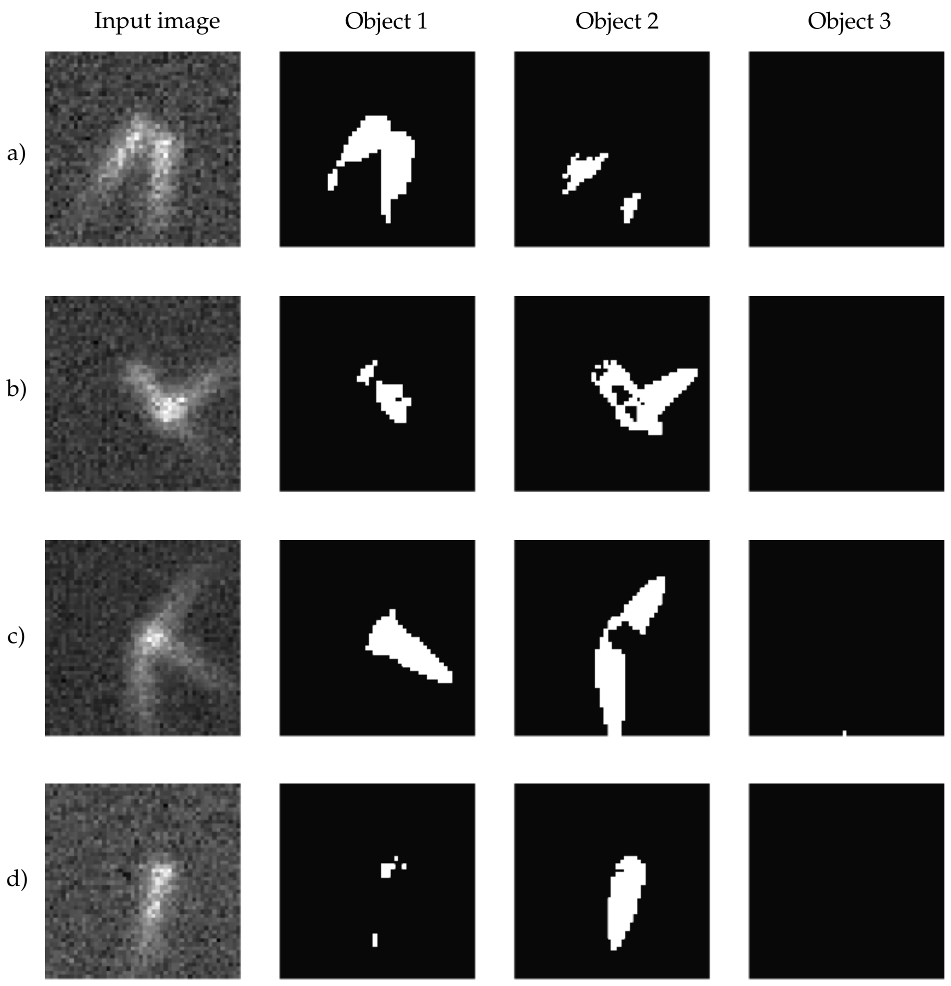

Figure 3.15: Qualitative results of the overlap network. The three objects represent the three classes that are predicted by the network. If the network would work perfectly, every microtubule segmentation would occupy a single image. The loss weights can be found in 3.4.

Table 3.4: Loss weights and learning rate for the overlap network in figure 3.15

Loss	Value
learning rate	0.001
Dice loss	0
Weighted CEL	5
MSE sum	500
CEL overlap	10
SensSpec	100
FTL	1000
Maximum projection	100
BCE semantic	100

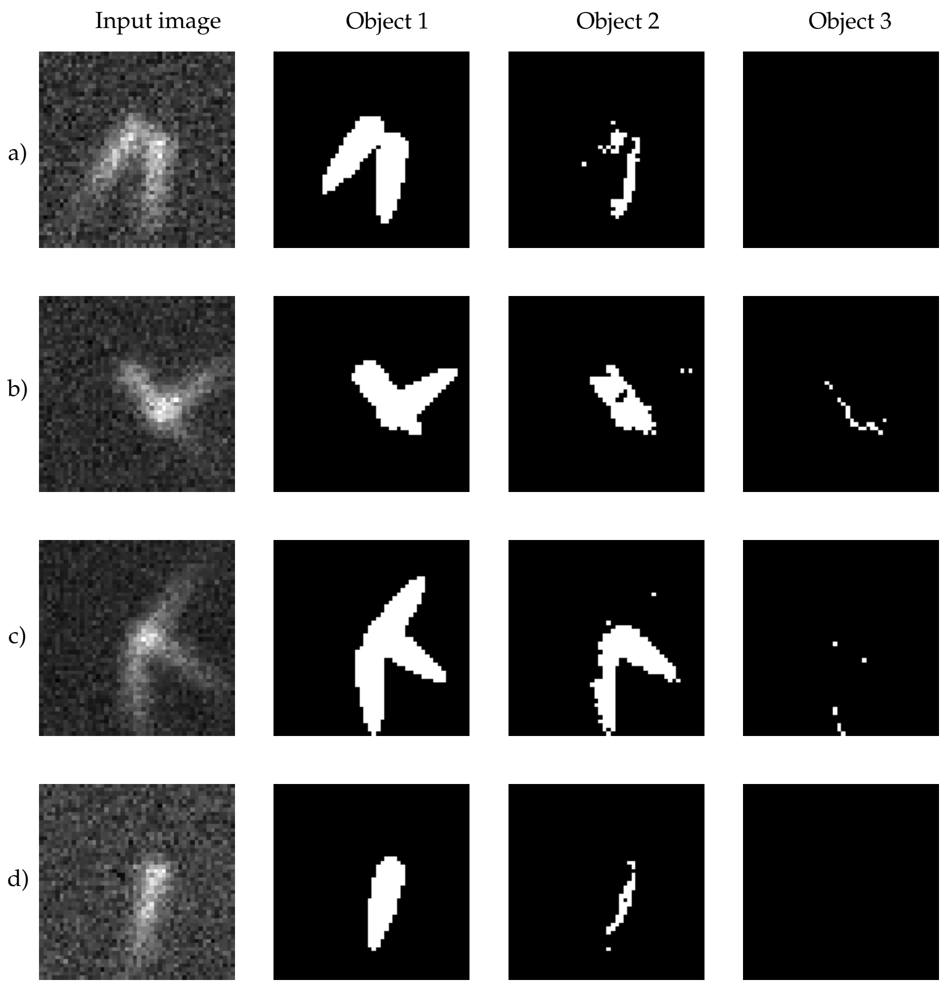

Figure 3.16: Qualitative results of the overlap network with the same weights as in figure 3.13, but trained on 1000 images.

Chapter 4
Discussion

In this chapter we discuss the results from chapter 3. We start by analyzing the outcomes of experiment 1 and experiment 2. Furthermore, we shed light on some limitations of the performed experiments. Following that, we explore the implications of these findings in the context of future research possibilities related to the topics addressed in this project.

4.1. Experiment 1

In figure 3.6 we can see the FROC curves and MSE of detections of both the U-net and ISOO$_{DL}$ network. In the FROC curve we can see that the ISOO$_{DL}$ network is able to outperform the U-net network. We also see that the ISOO$_{DL}$ network is able to reach a much higher FPR* than the U-net network. This is probably due to the fact that at low detection thresholds, all microtubule segmentations start to merge. This renders the network unable to produce many different detections, as the centers of mass of the segmentations are detections. Here we can see why it is useful to use reference points to produce detections. The ISOO$_{DL}$ network can produce more detections at low detection thresholds, because the reference points merge less quickly. However, the MSE of detections is lower for the U-net network.

The ROC curves for the segmentation of full images can be seen in figure 3.4. Here we can see that the U-net network outperforms the ISOO$_{DL}$ network. If we look at the JSC curve in figure 3.4. we can see that higher detection thresholds have higher JSC values than lower detection thresholds. Furthermore, we also see that U-net outperforms the ISOO$_{DL}$ network at all segmentation thresholds. We also see a difference between the shape of the U-net and ISOO$_{DL}$ JSC curves. The U-net curve is very broad and there is little fluctuation in JSC values. No matter what segmentation threshold is used, the JSC will not change that much. This means that the unthresholded segmentation output has a very sigmoidal characteristic, pixel values are either very close to 0 or very close to 1. In contrast to the U-net JSC curves, the ISOO$_{DL}$ JSC curves are less broad, and have a more gradual drop-off at higher threshold values. This means that the unthresholded segmentation output has a less sigmoidal characteristic, and pixel values are spread out more evenly between 0 and 1.

This behaviour could be explained by the weighing with a Gaussian in the post-processing steps of the ISOO$_{DL}$ network. To illustrate this, imagine we have a near perfect prediction of sheared masks. After applying our deshearing operation, we will get a prediction of almost 1 in the shape of a microtubule at the z=11 plane. All other pixels will be 0. By weighing with a Gaussian along the z-dimension, we are effectively reducing the contribution of the z=11 plane on the final segmentation. In our case, the z=11 plane will be weighed with a value of around 0.4. In the JSC curves of the detected boxes we can see that the optimal segmentation threshold is also around 0.4 (see figure 3.5). Therefore, we repeated our experiments with the σ of the Gaussian set to 0.5. Since the weight of the z=11 plane would now be 0.8, we expect the optimal threshold to shift to around 0.8. In figures 3.7 and 3.8 we can see the results of this experiment. Although we do see a change in the width of the JSC curve, the optimal segmentation threshold has not shifted to 0.8. This change in width can be explained due to the higher weighing of the z=11 plane. This plane should have the largest predicted values, and we are weighing those values with 0.8 instead of 0.4. This means that at higher segmentation thresholds, more pixels will be positively classified. However, as there is no shift in optimal segmentation threshold, the predicted values at z=11 will be lower than 1. This shows us that the desheared but unweighted masks have a less sigmoidal characteristic. It would be very interesting to examine if a different Gaussian could lead to more a sigmoidal characteristic and perhaps better segmentation performance.

Furthermore, we can also see that there is a shift in optimal segmentation threshold value for different detection thresholds. At higher detection thresholds, the optimal segmentation threshold is also higher. One explanation for this is that at lower detection thresholds predicted reference points in close proximity will merge. This will lead to detections that have a relatively high MSE, which is also something that we see in figure 3.6. Furthermore, there will also be more detections in locations that are not near reference points, which we can see in figure 3.11. This will also increase the MSE. The post-processing steps applied to the sheared network outputs rely on the location of detections. This means that even if the predicted sheared masks are correct, application of post-processing steps in the wrong location still leads to poor segmentation and thus a shift in optimal segmentation threshold.

Therefore, we also made ROC curves and JSC curves using the ground truth locations and the predicted sheared masks. Here we can see that although the ground truth detections lead to better performance compared to the predicted detections, the performance is still worse than U-net on a full image (figure 3.4). One might expect the ground truth detections to be able to reach a FPR of 1. However, in order to get a FPR of 1 all background pixels in the image have to be positively classified. As the bounding boxes still do not cover the entire image, an FPR of 1 is not achievable. We also see that the shape of the JSC curve is still less broad than U-net, suggesting that this is something inherent to the predicted sheared masks (figure 3.4). We see the same behaviour in the JSC curve of only the detected boxes (figure 3.5), which further reinforces our theory that the predicted sheared masks are less sigmoidal in nature. However, as discussed above this could also be partially attributed to the Gaussian weighing in the post-processing operations. Furthermore, we see little improvement in the ROC curve of only the detected boxes (figure 3.5). This curve is unable to account for any FN detection made. Since the ROC curve of the full image using ground truth detection does show an improvement, we can conclude that the ISOO_{DL} network makes too few detections. This also explains why the maximum achievable TPR and FPR of the ISOO_{DL} network are lower.

The exact reason why this shift occurs would require more experiments. However, these results do show us that if the detection threshold is changed, the segmentation threshold should also change for optimal results.

Although it might seem from these results that the ISOO_{DL} network is only a better detector of microtubules than the U-net network, the assignment of segmentations to specific instances is invaluable information. In figure 3.9 we can see that the ISOO_{DL} network is able to segment overlapping instances of microtubules in some cases, and also mark these as separate segmentations. In figure 3.10 we can see that even though U-net is able to detect that multiple instances of microtubules are present, it is unable to assign segmentations. Therefore, the ISOO_{DL} is already a big improvement compared to the U-net network, even though the segmentation results are worse.

4.2. Experiment 2

Figures 3.12, 3.13, 3.14, and 3.15 display the predicted segmentation results of four images using the overlap network. Although the network structure remains consistent across all figures, the weighing of losses were varied. The loss weights can be found in tables 3.1, 3.2, 3.3 and 3.4.

As we can see, the separation of microtubules along the 3 channels has proven to be very challenging. It is clear that the network is able to detect which pixels should belong to the background class, but the microtubule segmentations are fragmented across the different channels. One explanation for this could be that the weights used for the loss functions were not correct. Given the small difference in behaviour between the different figures we believe that the problem lies in the data. In the ISOO_{DL} data all pixels that belong to a microtubule should be put in a class. If you are looking at the xz-sheared predictions, the pixels of a microtubule with low x values should be in the bottom most class, and pixels with high x values should be in the topmost class. This way of structuring the data is independent of the amount of microtubules in the input image, and also independent of where other microtubules are classified. In contrast, the data for the overlap network does have dependencies between different microtubules. If one microtubule is assigned to one channel, then the other microtubules should be in other channels. However, there is no information in the input data that reveals in which channel a microtubule segmentation should be. Not only does the network have to correctly segment individual microtubules, it also has to assign the complete segmentation to a single channel.

Furthermore, we also see that the network is unable to assign microtubules to the third channel. One might argue that this is due to lack of exposure to three microtubules in the input image. However, we also trained the network on larger datasets (1000 images) and got worse results 3.16. We can see that at the network performs a semantic segmentation in the first channels of the output, and is unable to separate the microtubules. There seems to be an inability for all networks to predict anything in the third channel.

4.3. Future work

Based on the findings in this project there are quite a few ways in which this problem could be explored further. Therefore, this section will explore the potential avenues for further research beyond the scope of the present study.

4.3.1. Modifications to experiments

As was also discussed in 4.1 more experiments could be done to uncover why the ISOO_{DL} network produces

More experiments could be done to further optimize both detection and segmentation of the ISOO_{DL} network. As was discussed in 4.1 the segmentation of microtubules is still not optimal. Further experiments examine the effect of the post-processing operations on segmentation results, in particular the Gaussian weighing of desheared masks. Furthermore, the experiments could also be replicated for different SNR values, to see how our networks perform in less optimal situations. On top of that our network was trained and evaluated on synthetic data. In contrast to real data, synthetic data is less saturated to the highest intensity. This means that it is easier to detect overlap between microtubules, and thus our findings may not accurately reflect real-world applications. For a more realistic perspective, it would be ideal to replicate our experiments using real annotated data.

Additionally, it would be interesting to compare the ISOO_{DL} network to other region based networks, like Mask R-CNN or a modified YOLO. This would give us more insight regarding both the detection and segmentation performance.

4.3.2. DeepImageJ

Although the $ISOO_{DL}$ network still has more room for improvement, it would be useful to create an implementation using the deepImageJ environment. This would allow researchers to set thresholds for both detection and segmentation themselves, to suit their specific datasets. Therefore, it would also be nice to determine what combination of threshold values leads to optimal detection and segmentation results for more a more detailed range of values than was done in our experiments.

4.3.3. Variational Autoencoders

The main problem that we faced in this project was how to represent the data in such a way that the microtubules are separated, and that the network is able to reproduce this separation. The approach we have taken is based upon a pixel-wise classification of the input image. Another way to represent the microtubules would be via a small set of numbers that represent the location, shape and orientation of microtubules. This set of numbers could be predefined, but these can also be learned using a *Variational Autoencoder (VAE)*. A VAE is an autoencoder with an encoding and decoding pathway. The distribution in the encoding pathway are regularised, which means that the are restricted in some way. In our case this regularization would mean that the encoder produces the small set of numbers that describe the microtubule mentioned previously. The decoder tries to reproduce the input image using this small set of numbers. The decoder of the VAE can now be used to interpret small sets of numbers created by other networks as well. Therefore, another network can be created that takes an image of overlapping microtubules, and produces multiple sets that describe these microtubules. By feeding these sets one by one through the decoder of the VAE, multiple images with only one microtubule in each will be created. On top of that the VAE does not take the set of number from the encoder, but it draws numbers from a distribution based on this set of numbers. This introduces some variance into the encoder. We hope that this leads to the ability to change one characteristic of a microtubule by changing one of the numbers in the set.

4.3.4. Discriminator

Another way to increase performance of the proposed networks in this thesis could be using concepts from *Generative Adverserial Networks (GANs)*. A GAN network structure has two main tasks: generation and discrimination. The first part of the network is a data generator that tries to emulate the data distribution of the input data. The discriminator takes samples from the generator and 'true' data and tries to classify them. The main idea of a GAN network is that you try to fool the discriminator by training the generator. We could take this concept and apply it to one of the networks we trained in this thesis. One problem we faced with the overlap network in section 2.2.3 is that it produces fragmented microtubules in each of the output channels. It is very difficult to define a loss function that is able to penalize this fragmentation. However, a discriminator is a trained network, and the loss function would not have to be predefined. A discriminator could thus be used as an additional loss during the training of the network.

Chapter 5
Conclusion

The primary objective of this project was to automate the detection and segmentation of microtubules using deep learning models. In section 1.5 we outlined a set of research questions, which we will attempt to answer in this section.

The ISOO_{DL} network consists of a fully convolutional neural network, in combination with some post-processing steps. This network predicts *reference points*, defined as small disks at the center of mass of microtubules. These reference points are smaller than microtubules, and are allowed to shrink if in the vicinity of other reference points. Therefore, overlapping microtubules will still have separated reference points. After post-processing operations, the predicted reference points can be converted to coordinates. Furthermore, the network predicts sheared segmentation masks in both the xz and yz direction. In this 3 dimensional space, all microtubules are separated. A small bounding box is placed at every predicted coordinate, and deshearing operations are applied. This results in a unique segmentation mask for every detected microtubule.

We have shown that the ISOO_{DL} network is able to detect and segment microtubules. Although the detection of microtubules is better than a regular U-net structure, the segmentation is still an area for improvement. We saw that the maximum TPR changes depending on the detection threshold. The maximum TPR using ground truth detections was significantly better, which tells us that improving the detection of microtubules also leads to better segmentation results. From the shape of the JSC curve, we could deduct that the predicted segmentations do not have a very sigmoidal characteristic. This can partially be attributed to the post-processing operations. Further experiments could shed light on how the post-processing operations can be optimized, to increase segmentation performance. We have also seen that the optimal segmentation threshold changes depending on the detection threshold. By visually inspecting the predicted segmentations, we could see that overlapping microtubules were more challenging to segment.

In hopes of further improving segmentation of overlapping microtubules, we proposed a new network. We found that the ISOO_{DL} network was able to predict the amount of overlap present in an image. Therefore, we used the amount of overlap together with the raw data as inputs to this 'overlap network'. This network attempts to place microtubule segmentations in three separate channels of the output. Unfortunately, we found that this way of data separation was unsuccessful. This is probably because there is no information in the input data regarding which microtubule should be placed in what channel.

In conclusion, we can detect and segment isolated microtubules. However, the detection of overlapping microtubules is more challenging. Additionally, the segmentation of overlapping microtubules still requires further research. Nevertheless, we have shown that segmentation of overlapping microtubules is to a limited extent possible using the ISOO_{DL} network. Other network structures could result in further segmentation improvements.

Bibliography

[1] Anton Böhm et al. “Isoo dl: Instance segmentation of overlapping biological objects using deep learning”. In: *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*. IEEE. 2018, pp. 1225–1229.

[2] Arshad Desai and Timothy J Mitchison. “Microtubule polymerization dynamics”. In: *Annual review of cell and developmental biology* 13.1 (1997), pp. 83–117.

[3] Tatiana Stepanova et al. “Visualization of microtubule growth in cultured neurons via the use of EB3-GFP (end-binding protein 3-green fluorescent protein)”. In: *Journal of Neuroscience* 23.7 (2003), pp. 2655–2664.

[4] Jan Marc et al. “A GFP–MAP4 reporter gene for visualizing cortical microtubule rearrangements in living epidermal cells”. In: *The Plant Cell* 10.11 (1998), pp. 1927–1939.

[5] Erik Meijering et al. “Imagining the future of bioimage analysis”. In: *Nature biotechnology* 34.12 (2016), pp. 1250–1255.

[6] Sebastian P Maurer et al. “EB1 accelerates two conformational transitions important for microtubule maturation and dynamics”. In: *Current Biology* 24.4 (2014), pp. 372–384.

[7] Kaiming He et al. “Mask r-cnn”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

[8] Nicolas Chenouard et al. “Objective comparison of particle tracking methods”. In: *Nature methods* 11.3 (2014), pp. 281–289.

[9] Estibaliz Gómez-de-Mariscal et al. “DeepImageJ: A user-friendly environment to run deep learning models in ImageJ”. In: *Nature Methods* 18.10 (2021), pp. 1192–1195.

[10] Caroline A Schneider, Wayne S Rasband, and Kevin W Eliceiri. “NIH Image to ImageJ: 25 years of image analysis”. In: *Nature methods* 9.7 (2012), pp. 671–675.

[11] Nasim Jamali et al. “2020 BioImage Analysis Survey: Community experiences and needs for the future”. In: *Biological imaging* 1 (2021), e4.

[12] Adam Paszke et al. “Automatic differentiation in PyTorch”. In: *NIPS-W*. 2017.

[13] Wei Ouyang et al. “Bioimage model zoo: a community-driven resource for accessible deep learning in bioimage analysis”. In: *bioRxiv* (2022), pp. 2022–06.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[15] Augustin Cauchy et al. “Méthode générale pour la résolution des systemes d’équations simultanées”. In: *Comp. Rend. Sci. Paris* 25.1847 (1847), pp. 536–538.

[16] Yann N Dauphin et al. “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization”. In: *Advances in neural information processing systems* 27 (2014).

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

[18] Tijmen Tieleman, Geoffrey Hinton, et al. “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude”. In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.

[19] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. pmlr. 2015, pp. 448–456.

[20] Shibani Santurkar et al. “How does batch normalization help optimization?” In: *Advances in neural information processing systems* 31 (2018).

[21] Yi-Tong Zhou and Rama Chellappa. “Computation of optical flow using a neural network.” In: *ICNN*. 1988, pp. 71–78.

[22] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer. 2015, pp. 234–241.

[23] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. “Rectifier nonlinearities improve neural network acoustic models”. In: *Proc. icml*. Vol. 30. 1. Atlanta, Georgia, USA. 2013, p. 3.

[24] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[25] Anton Böhm, Maxim Tatarchenko, and Thorsten Falk. “Isoo v2 dl-semantic instance segmentation of touching and overlapping objects”. In: *2019 IEEE 16th International symposium on biomedical imaging (ISBI 2019)*. IEEE. 2019, pp. 343–347.

[26] Nobuyuki Otsu. “A threshold selection method from gray-level histograms”. In: *IEEE transactions on systems, man, and cybernetics* 9.1 (1979), pp. 62–66.

[27] Thorvald A Sorensen. “A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons”. In: *Biol. Skar.* 5 (1948), pp. 1–34.

[28] Lee R Dice. “Measures of the amount of ecologic association between species”. In: *Ecology* 26.3 (1945), pp. 297–302.

[29] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. “V-net: Fully convolutional neural networks for volumetric medical image segmentation”. In: *2016 fourth international conference on 3D vision (3DV)*. Ieee. 2016, pp. 565–571.

[30] James Munkres. “Algorithms for the assignment and transportation problems”. In: *Journal of the society for industrial and applied mathematics* 5.1 (1957), pp. 32–38.