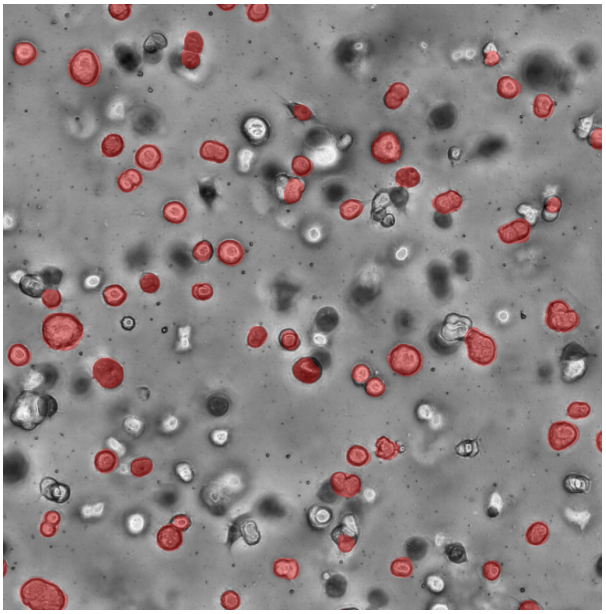
UNIVERSITY OF COPENHAGEN
DEPARTMENT OF COMPUTER SCIENCE



Bachelor Thesis

Niels Dam Fisker & Oscar Koch-Müller

Segmentation of cells in focus

A comparative analysis of machine learning and image analysis approaches

Date: June 2025

Advisors: Jon Sporring & Silja Heilmann

Contents

Abstract

This thesis examines various methods for image analysis to segment cells captured in a 3D-image stack containing brightfield images. The study examines two paths to make such segmentations. One approach combines image filters (Laplacian and Sobel) with mathematical morphology, while the other employs machine learning, specifically a U-Net using RootPainter. Ultimately, the goal is to create a method that performs better than an existing random forest, which is already trained for this purpose, but is computationally slower than desired. These methods were then compared on object level using Intersection over reference (IoR) and on pixel level. The results point to the methods having different strengths, as the image filters combined with morphology achieve higher recall scores, and the U-Net achieves higher precision scores with a better performance in the segmentation overall, as indicated by a higher F1 score. The thesis also comparatively analysed the use of 2D- and 3D-segmentations for estimating the 3rd dimension. The best method was using a 3D-segmentation for finding the 3rd dimension, but it could still be found, although with lesser accuracy from a 2D-segmentation.

Preface

This project aims to apply the knowledge we have gained throughout our studies of computer science to a real-world problem in collaboration with a Biotech startup company, which does not want to be named, in the report. We hope to support their innovative research in improving cancer treatment procedures by improving methods for the efficient detection of cells. The project description did include a possibility of doing object classification, but due to being occupied with object detection, we were unable to address the classification.
The reader is expected to have academic qualifications equivalent to a bachelor's degree in computer science and be familiar with machine learning and image analysis techniques.

All data used for tests and code can be found in the ERDA folder, with a user guide in appendix 15.3. A text file, Link_to_ERDA.txt, containing a link to the folder is handed in alongside this document.

1 Introduction

One of the biggest diseases in the world is cancer, and we get better and better at detecting and curing it year by year, but approximately 50-70% of patients don't respond at all to the first-line treatment. The company aims to develop a second or 3rd line treatment to assist oncologists in picking the best treatment when the first-line treatment fails. They aim to do this by analysing which medicine is best to stop the spread of the cancer in the patient. This is to be done by taking cells from the patient by core needle biopsy from a metastatic lesion, exposing them to different types of medicine, and analysing their responses. Since they are only starting to solve their problem, most of their analysis pipeline is not finished. All this analysis must be done automatically, as the workload will be too expensive and impractical to do manually, due to the high amount of man-hours required. In this project, we will work on an early part of the pipeline, as it is shown in Figure 1, which involves segmenting the images of the cells and extracting the areas that are needed for the analysis, which are areas containing objects that are in focus. It should be noted that no ground-truth labels were provided, which limits the possible approaches to tackle this problem.

Figure 1: Data pipeline of the company

In their current approach, they have used a random forest model trained in the program Ilastik (Ilastik, 2025), which showed satisfying results, but the segmentation of one stack of images takes approximately 40 min. They want to speed this process up because this approach is too slow and costly if their method is to be used for many cancer patients, according to the company. As a method of solving this, they want to use the deep learning model YOLO from Ultralytics and have started to experiment with it. YOLO is a fast model and shows good results in different datasets, which is an important reason in their choice of model (Ultralytics, 2025). In our analysis of this choice, we see some potential issues with their approach to using the model, which we will now describe: The images of the cells are contained in a 3D-stack, and YOLO takes 2D-images with 3 colour channels. So, to solve this problem, the company has been projecting the 3D-stack down into 2D-images using a standard deviation, mean, and max projection in the height axis, and passing this projection to the model in the colour channels of a 2D-image. This will result in a loss of information, as the location of the cells can only be found in two out of three axes by the model. This becomes relevant if the information in the third axis is to be used in further analysis and cannot be found reliably in another way. It is also unknown how well the model handles this projection, as the relationship between the three channels is different from normal RGB color channels, and whether projections capture relevant information from the axis they represent, such that they can find what they want in the images.

A dataset with images and corresponding annotations was needed to train their YOLO model. Because of time restraints in man-hours, they have been using their slow model trained in Ilastik to generate annotations, to be used for the training of the YOLO model, without analysing the performance of their Ilastik model, which can lead to errors in the YOLO model. That is because the YOLO model may be given a lot of false information to learn from, due to errors in the Ilastik model, which can hurt the model's performance and therefore be a bad choice, as indicated by previous research (Koksal et al., 2020). Because of this, we will ask the question: Is YOLO the right choice given the reasons above, or are there alternative ways to accomplish this?

Our project:

We will try to find alternative ways of segmenting the images to locate the relevant objects in the images without using YOLO. In doing this, we will also try to answer whether the important information from the axis they project down from 3D to 2D can be easily retrieved afterwards, or if a segmentation in 3D is needed. Our criteria for working with this problem: Our models must produce segmentations of the objects in the images with good performance, and be able to give the placement in all three dimensions. Our ways must be faster than their Ilastik model.

2 Random forest

The company provided us with their segmentations of all the stacks, from a model trained in the segmentation tool Ilastik. It uses a random forest model, which gets the input file from the user and applies image processing filters such as Laplacian and Sobel, to make a model that predicts annotations made by the user. The exact details of how the model was trained and created are unknown to us, but we do know that many filters and processing techniques were applied to the input images, which is a time-consuming process, and one stack took approximately 40 minutes to segment. This runtime was not satisfactory, and we have therefore sought alternative methods that can still segment as well, but can improve the runtime. We are aware that being able to conclude improved runtimes definitively would require us to run the test on the same hardware or get the model to run on our computer, but since this was not feasible, we've chosen to test it to run times on one of our computers, which was slower than their computer.

3 Data

In this section, we will analyse and describe the data that we will be working with in this project.

3.1 Creation of the data

The data used in the project consists of 3D-stacks with images of cancer cell cultures, taken with BioSense Solutions ocelloscope. The images are captured at three time points, after 1, 6 and 11 days of growth. The microscope used is a brightfield microscope, which has a light source above and a camera below the well in which the cells are located. Brightfield microscopes have a very narrow field of view, so to capture objects clearly, they need to be in the right place; otherwise, the objects quickly become blurred in the image. The images are captured using BioSense Solutions' FluidScope scanning method. During the scanning, the microscope sits at a 6.25-degree angle to the bottom plane of the well, and it uses a single motor to move over the well, where it takes images at a fixed interval as shown in Figure 2 (Fredborg et al., 2013). These images are stacked on top of each other to produce the image stacks that form the basis for this project.

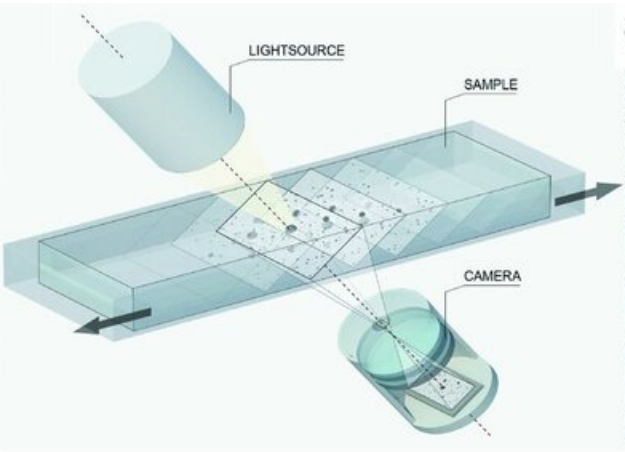


Figure 2: Illustration of how the images are captured (Fredborg et al., 2013)

This means that the coordinate system of the well is not aligned with the coordinate system of the 3D-stack. For simplicity in the report, the coordinate system refers to those of the stack, unless stated otherwise.

3.2 3D-stack analysis

The images in the stack are padded with 0s so the xy-coordinates of each image remain the same in all layers of the stack, so that objects don't move when looking in different layers. This makes the stack very sparsely filled, which can be seen in Figure 3.

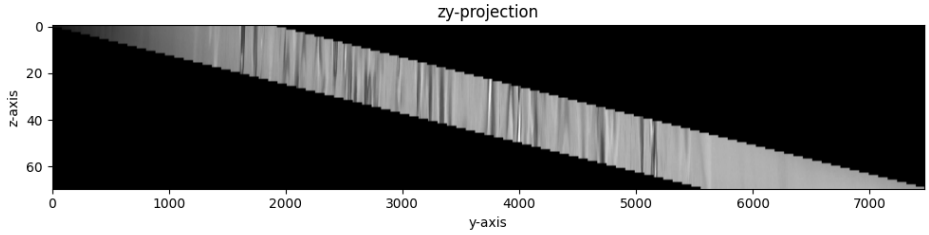


Figure 3: Image of a stack in zy-plane

Each 3D-stack consists of 70 layers of greyscale images of the cells, where each voxel is a uint8 value. Each layer is around 7460x2560, although it varies a little from stack to stack, which is due to the process of capturing the images. We were provided a total of 275 stacks. The vast majority of these stacks were used for the training set, while only a small portion was left for validation and test sets, as we would need to manually annotate these. The part of each layer where there is image data for a given layer is around 1980x2560; the rest of the layer is padded with 0s to align the cells in the xy-plane. This can be seen in Figure 4, which shows a layer in a stack.

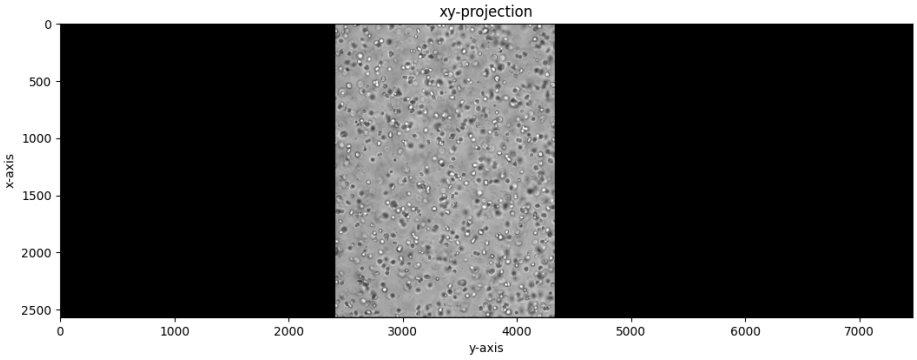


Figure 4: An image slice of the stack in xy-plane

The 3D-stacks are taken at fixed time points, so the growth of the cells can be analysed. This growth of the cells and the company adding a reagent makes the images from the later time points darker, which can be seen in the histograms of a full 3D-stack for the different time points in Figure 5 and from a layer from each point in Figure 6, which shows that point 6 and 11 are a lot darker, than point 1.

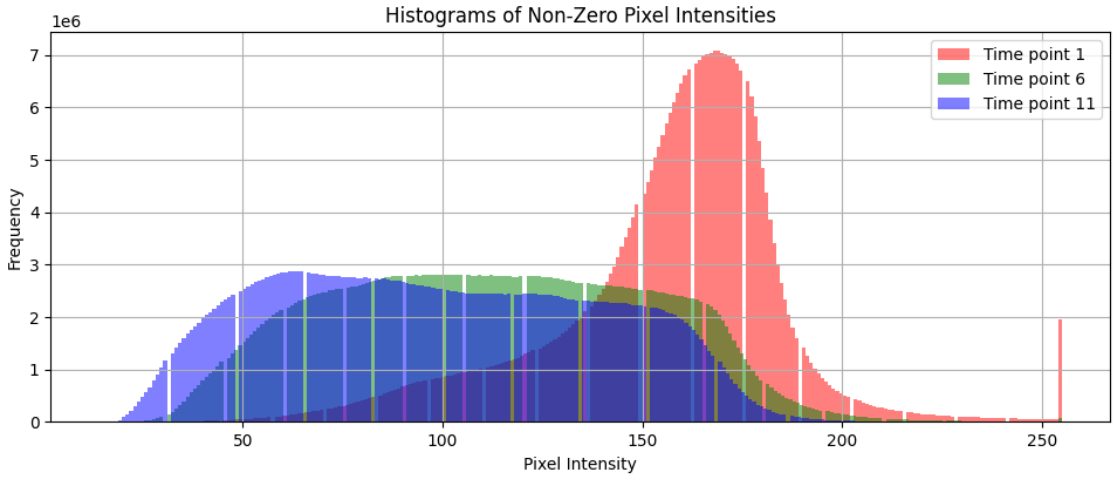


Figure 5: Histogram of one stack, at each time point

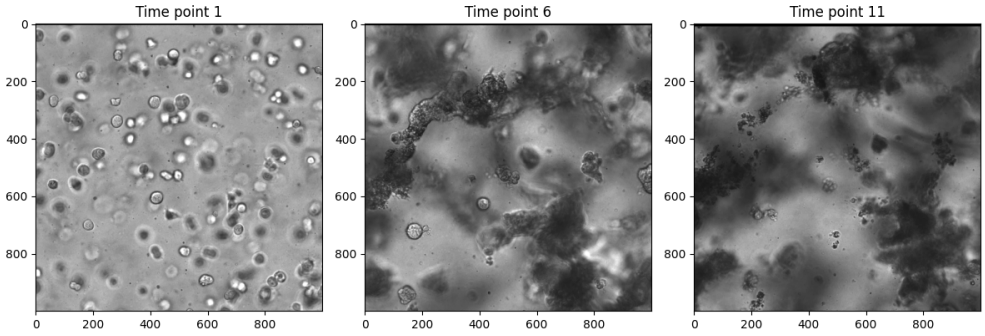


Figure 6: Image showing data at different time points

3.3 Visual image analysis of the cells

We are interested in finding all the objects that have an image where the object is in focus. Figure 7 shows a fixed cell in the different layers, as it comes in and out of focus. From that series, we see that when the focus plane of the microscope is below the cell in the z-axis, the image shows a darkened, blurred blob, and when the focus plane is above the cell, it becomes lighter and blurred. Somewhere between these two extremes, the cell appears to be in focus.

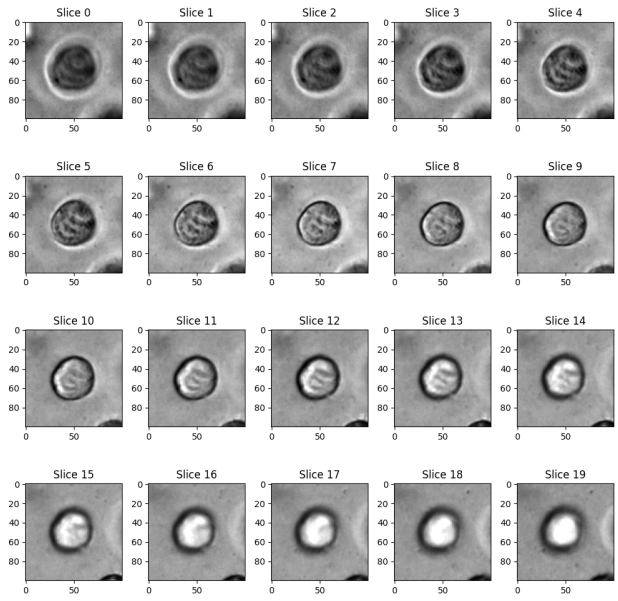


Figure 7: A cell shown in different layers

3.4 Scale of the z-axis

The use of brightfield microscopy has several implications for the z-axis of the images. First, the microscope takes images throughout the whole well, which means that all objects between the light source and the camera influence each image; an object is shown on all images, which contains its location, but is shown in a different amount of focus. Second, the microscope has a narrow focus field, which implies that cells in the images are only in focus in a few layers, where the focus of the microscope aligns with the cell, but the cell can be seen in multiple layers. This means that finding the layer of best focus is important to finding the object's placement in 3D. Third, when we look at an image of the xz- or yz-plane of the stack, the depiction of a cell is different from the view in the xy-plane, as is shown in Figure 8. This comes from the shift of focus through the z-axis, and it will make the z-axis a different measure than that of the x- and y-axis. Where the x- and y-axis are measures of distance, the z-axis represents the relative position of the camera when taking the image and therefore how much the object is in focus, which can be used as a proxy measure of distance. This makes it difficult to analyse in 3D as the axes are not the same, which becomes an argument for splitting them up and analysing them separately in this project.

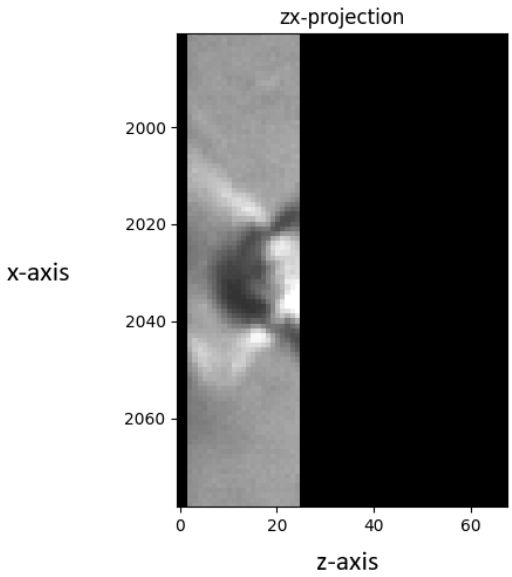


Figure 8: A cell shown in zx-plane

4 Evaluation measures

Due to the nature of the data, locating objects in the xy-plane and on the z-axis are two fundamentally different tasks, and we're therefore evaluating them separately. This section will concern itself with the xy-plane and how we evaluate segmentations on this plane, while section 8 will deal with the z-axis.
When deciding on what evaluation metrics to use when evaluating our results, we realised that most well-established metrics did not quite reflect what we wanted to evaluate. Since the project is primarily about detecting objects and getting exact segmentations on a pixel level is of low priority, one logical way to approach it would be to apply a matching algorithm followed by some criterion for evaluating whether the matched segmentation is similar enough to the object it is matched to. The issue with this approach is that often, the objects are located very close to each other, resulting in the segmentation connecting two or more objects. If we were to evaluate this with the approach described above, then one of the objects would be considered a match and the others misses, even though we did detect it. Figure 9 illustrates this issue.

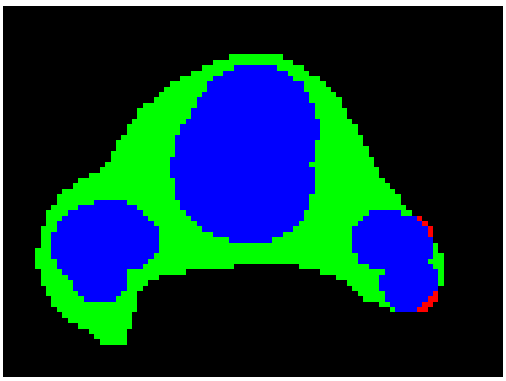


Figure 9: A snippet of a segmentation by the random forest applied to a stack at time point 1. Blue is true positive, black is true negative, red is false negative and green is false positive.

Using the approach above, at most one of the three objects from the annotation would be found here, as the segmentation only found one object, even though it mostly covers all three annotated objects. To handle

this, we decided to use intersection over reference (IoR) (Metrics Reloaded, 2025b) with both the annotations and segmentations as the reference object as well as a modified version of this, which, unlike IoR, allows multiple fractioned objects to count as a positive if their union covers enough of the reference object. We decided that for an object to be detected, at least 50% of it must be found. Now, given that we calculate this measure twice, once with annotations as reference and once with segmentations as reference, we can interpret the results as measures of precision and recall for a given image. These metrics are defined mathematically as follows:

Define A as the set of every annotation.
Define S as the set of every segmentation.
Assume $(S \neq \{\}) \wedge (A \neq \{\})$, Then:

$$\text{Fractioned IoR Precision} = \frac{\sum_{s \in S} \mathbf{1}\left[\sum_{a \in A} |s \cap a| \geq \frac{|s|}{2}\right]}{|S|} \tag{1}$$

$$\text{Fractioned IoR Recall} = \frac{\sum_{a \in A} \mathbf{1}\left[\sum_{s \in S} |a \cap s| \geq \frac{|a|}{2}\right]}{|A|} \tag{2}$$

$$\text{IoR Recall} = \frac{\sum_{a \in A} \mathbf{1}\left[\exists\, s \in S \mid |a \cap s| \geq \frac{|a|}{2}\right]}{|A|} \tag{3}$$

$$\text{IoR Precision} = \frac{\sum_{s \in S} \mathbf{1}\left[\exists\, a \in A \mid |s \cap a| \geq \frac{|s|}{2}\right]}{|S|} \tag{4}$$

If $(S \neq \emptyset) \wedge (A \neq \emptyset)$, then we set all of the measures defined in (1), (2), (3) and (4) to 1, as that must represent perfect prediction.
If $(S \neq \emptyset) \wedge (A = \emptyset)$, then we set (2) and (3) to 0 and (1) and (4) to 1, as that must mean perfect precision and the worst possible recall.
If $(S = \emptyset) \wedge (A \neq \emptyset)$, then we set (1) and (4) to 0 and (2) and (3) to 1, as that must mean perfect recall and the worst possible precision.

We don't expect a large difference between the fractioned and the unfractioned IoR measurements for early time points, as the objects tend to be rather small and clear. But in late time points, the objects can be large, which at times causes the segmentations to be fractioned, even though they cover the annotations sufficiently when combined. Ignoring this point and simply considering that as a miss wouldn't be true to the goal of this project, and therefore, we decided to use both measures.

With these definitions, the issue seen in Figure 9 is circumvented as (2) and (3) will both be 1, as 50% of every annotation is covered by a single segmentation. (1) will also just barely be 1, as there are 1203 blue pixels and 1131 green pixels, but if the model was to predict more false positives than true positives for the object, it would be 0. (4) will be 0, as 50% of the segmentation isn't covered by one single annotation.

Based on the above measures, we can calculate the F1 score as:

$$F_1 = 2\frac{\text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}} \tag{5}$$

Even though the exact segmentations aren't of great importance, we will still be calculating the pixel-wise F1 score to ensure the segmentations are not consistently very far off from the annotations in this regard.
In general, our focus will be on the fractioned F1 score, as this is the closest to what we want to evaluate. Now, it is possible to set up scenarios where this measure would be undeservedly high, and for that reason,

we will look at it in collaboration with the other measures mentioned.

For the measures, we will be looking at the per-sample values, i.e., the value is calculated for each sample, and then an average is taken across all samples. This is because we don't want time point 1 to dominate in the combined analysis, as it has the most elements. We also want the score to be equal if 50% of objects are found, regardless of there being 5 or 50.

5 Labeling ground truth

In this section, we will go through how we made our annotations for the validation and testing of our models. We need these annotations as we need some annotations to test our models against when we are selecting parameters in the validation phase, and to test the models against each other in the testing phase.

Before we began annotating, we set up some ground rules we should follow during the annotation:
We draw all of the areas of the objects that are in focus at some point in the 70 layers.
If an element is overlapped by a shadow from another cell, but is in focus, it is still annotated.
An object is only annotated in the layer of the most focus. For the cell from Figure 7, it would be in slice 8 that cell would be annotated.
Then we started to annotate, we annotated some elements together for each time point, to get a united agreement on how an element in focus is represented in the stack. After that, we talked about some edge cases where the cell is at the edge of the stack and whether they were in focus or not, as we don't have the "out-of-focus, in-focus, out-of-focus" structure to go by (as in figure 7) if they're located at the very edge.

For deciding which stacks to annotate, we took random stacks as chosen by random.org's (Random.org, 2025) generator, then sampled a random 1000x1000 patch, while keeping the entire z-axis, resulting in samples of size 70x1000x1000 voxels. We ensured that the samples were from the same areas across time points, i.e., for a given stack where we sampled a certain area, that same area would also be taken up for annotation in the stacks representing other time points for the same cells. Then the patches were divided between us, so we each did half from each time point. The patch size was chosen to give a balance between the number of elements that are represented and the number of voxels, at all time points. This is needed as the objects at time points 6 and 11 are a lot bigger than those at time point 1.
For the validation set, we chose 5 random stacks at the 3 time points, i.e., a total of 15 patches. After we were finished with the evaluation of the models on the validation set, we annotated the test set from 4 random stacks at the 3 time points, i.e., a total of 12 patches.
Figure 10 shows an example annotation for each time point. Do notice that some objects that looks like they are in focus are excluded, as we decided they were in better focus in another nearby layer.

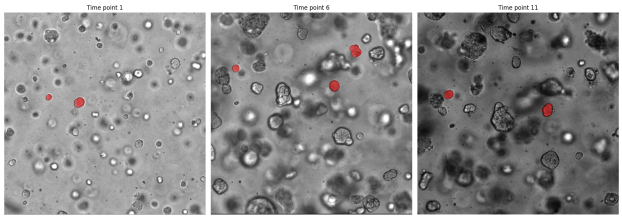


Figure 10: Example slices from annotations, one from each time point

6 Segmentation using mathematical morphology

We built segmentation models based on mathematical morphology. We will now discuss the various approaches we explored to do this.

The morphological operations we applied are defined as follows:

For binary morphological operations:
Define $\Omega \subseteq \mathbb{Z}^n$ as an integer lattice.
Define $A \subseteq \Omega$ as the binary image.
Define $B \subseteq \Omega$ as the binary structuring element.
Define $A + b$ as the translation of A by b, i.e., $A + b = \{a + b | a \in A\}$.
Then:

$$\text{Binary dilation of } A \text{ by } B: \quad A \oplus B = \bigcup_{b \in B} A - b \tag{6}$$

$$\text{Binary erosion of } A \text{ by } B: \quad A \ominus B = \{\omega \in \Omega | B + \omega \subseteq A\} \tag{7}$$

Now we can define the binary opening and closing operations as:

$$\text{Binary Opening of } A \text{ by } B: \quad A \circ B = (A \ominus B) \oplus B \tag{8}$$

$$\text{Binary Closing of } A \text{ by } B: \quad A \bullet B = (A \oplus B) \ominus B \tag{9}$$

For greyscale morphological operations:
Define $\Omega \subseteq \mathbb{Z}^n$ as an integer lattice.
Define $A : \Omega \to \mathbb{Z}$ as a greyscale image.
Define $B : \Omega \to \mathbb{Z}$ as a greyscale image.
Then:

$$\text{Greyscale dilation of } A \text{ by } B: \quad (A \oplus B)(x) = \max_{y \in \text{Supp}(B)} [A(x - y) + B(y)] \tag{10}$$

$$\text{Greyscale erosion of } A \text{ by } B: \quad (A \ominus B)(x) = \min_{y \in \text{Supp}(B)} [A(x + y) - B(y)] \tag{11}$$

Greyscale opening and closing are defined analogously to those for binary images.

We constructed models for 2D-projections as well as for the 3D-data itself to evaluate which approach is better performing. In general, we worked on three different approaches to segmentation using mathematical morphology:

- 2D-projection, followed by morphological operations
- Derivative filter followed by morphological operations
- Derivative filter followed by morphological operations on every layer in the z-direction

By derivative filter, we mean either a Sobel or a Laplacian filter. We've used both independently, but the approach is the same no matter which filter we applied. We took the absolute values after applying the filters, as we're just interested in the magnitude.
In common for all of these approaches is that our general approach on deciding what parameters to use was a mix of trial and error and evaluating it against our validation set; we would look at segmentations produced by a given model and then try to adjust the parameters, such that the segmentations produced by the new model adressed the deficiencies of the previous iteration.
We will discuss each of these approaches in the following.

6.1 Morphology on a 2D-projection

This subsection will discuss our approach to creating morphology-based models on 2D-projections.

6.1.1 Projecting the data to 2D

By projecting the data to 2D, we were hoping for a speed up in runtime, as it would be much faster in terms of morphological operations. Of course, the cost of actually projecting the data to 2D is a cost that should be included in this calculation.
To apply morphology to a 2D-projection, we would, naturally, first need to project the data to 2D. We do this by applying a weight function along the z-axis. Now, there exist many reasonable weight functions to perform this projection. We experimented with many such functions, which we deemed reasonable and applied them all to the same 3D-stack. The results of the application of the weight functions to one stack can be seen in Figures 11 to 25. Due to the size of these, we only display a crop out of the full projections, such that it is easier to visually understand what is going on. The full projections can be found in appendix 15.1. There is no qualitative way to decide which weight functions produce the best results, and therefore, we had to choose based purely on quantitative characteristics. The two projections that we continued to work on were the weighted Laplacian and weighted Sobel projections, Figures 20 and 21, as we deemed them to quite faithfully represent the 3D-data. We were especially fond of the weighted Laplacian projection and believe it is the 2D-projection that most accurately depicts the 3D-data from the ones we explored.

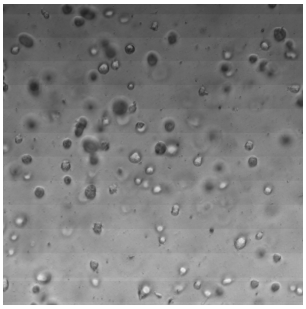


Figure 11: Mean projection

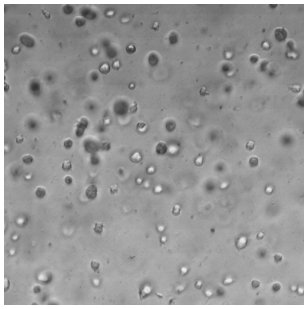


Figure 12: Standard deviation projection

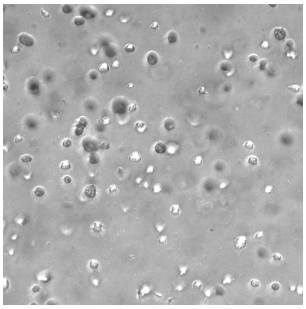


Figure 13: Max projection

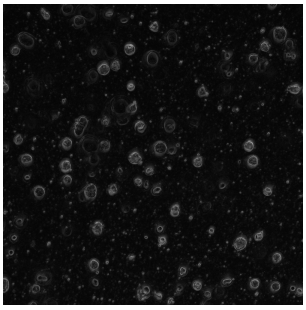


Figure 14: Max intensity projection after application of a Sobel filter

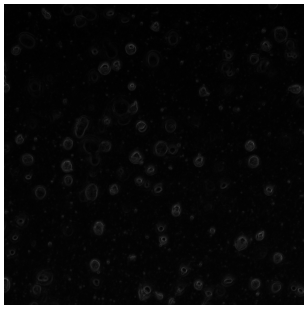


Figure 15: Mean intensity projection after application of a Sobel filter

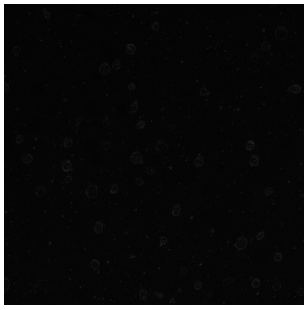


Figure 16: Max intensity projection after application of a Laplacian filter

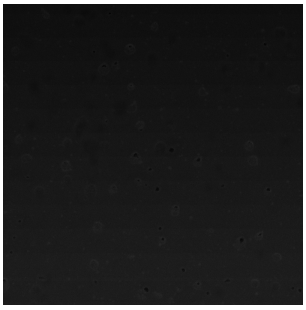


Figure 17: Mean intensity projection after application of a Laplacian filter

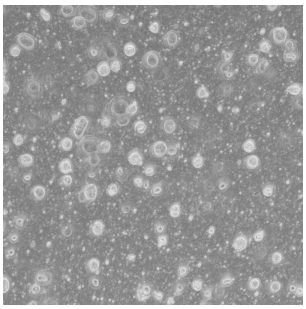


Figure 18: Max intensity projection after application of a Sobel filter on a logarithmic scale

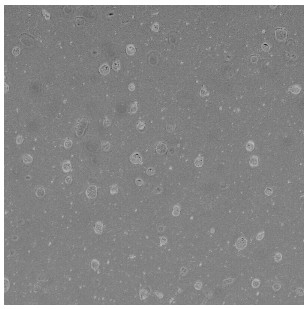


Figure 19: Max intensity projection after application of a Laplacian filter on a logarithmic scale

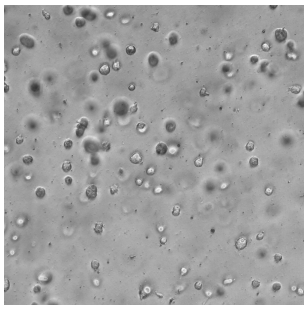


Figure 20: Application of a Laplacian filter followed by a weighted sum of the original image based on Laplacian magnitude

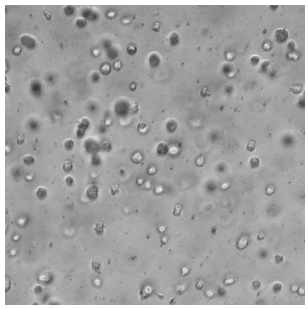


Figure 21: Application of a Sobel filter followed by a weighted sum of the original image based on Sobel magnitude

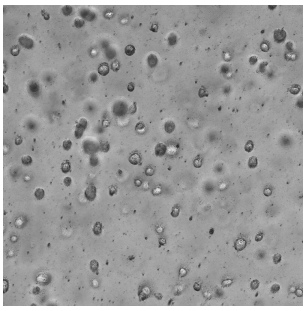


Figure 22: Application of a Laplacian filter, followed by taking the original value of the layer with the highest Laplacian value

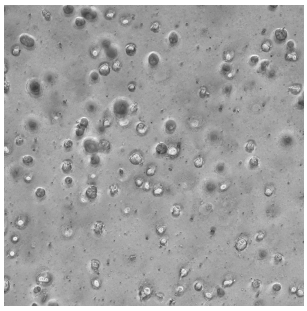


Figure 23: Application of a Sobel filter, followed by taking the original value of the layer with the highest Sobel magnitude

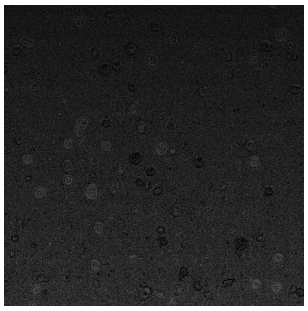


Figure 24: The layer with the highest Laplacian value

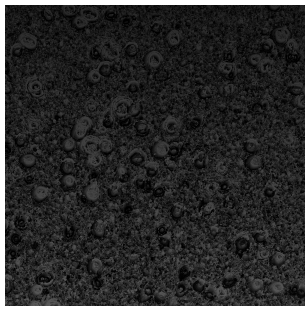


Figure 25: The layer with the highest Sobel magnitude

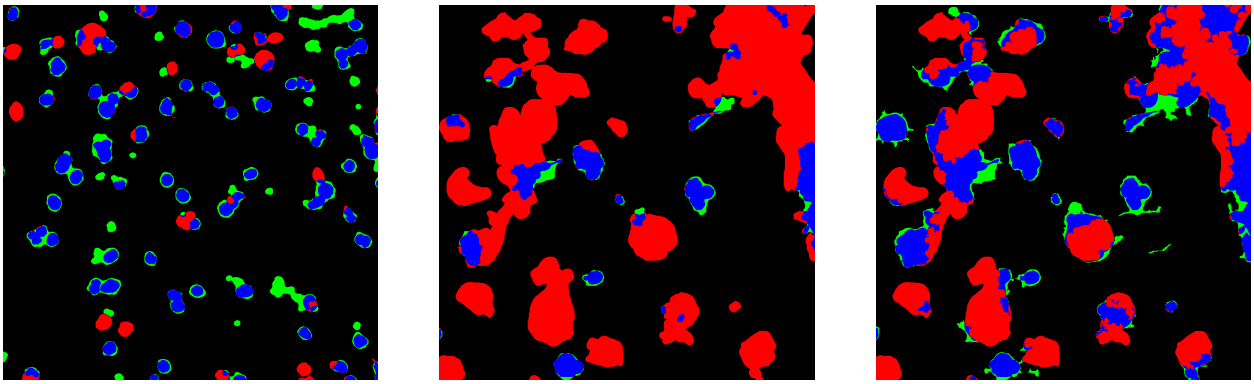
6.1.2 Applying morphology

For this approach, we first applied a 2D-projection as discussed in section 6.1.1, and we used both the weighted Laplacian and Weighted Sobel projections for this. We used the values after application of the filter to gauge how sharply the pixel values change, and we use this change to measure whether a given object is in focus.
We then used various greyscale morphological operations on the images. In general, the morphology was designed to close holes in elements and remove elements in the mask that are a result of noise. We followed this with a threshold, and in some cases, we also applied binary morphological operations after the threshold. We varied the 2D-projection used, the greyscale morphology, the threshold and the binary morphology in the cases where it was applied.

We optimised a model of this kind for all of the data points and one for each of the three time points individually. In total, this led to 31 different sets of parameters for the 2D-morphology approach.
After validating the models on our validation set, to find the best model for each time point and the best one across time points, we ended up with only two models, as the best performing across all time points also happened to be the best one for time points 6 and 11. The two models are in general very similar, they both use a weighted Sobel projection (see Figure 21 for an illustration of this projection), whereto we apply a greyscale closing with a circular, binary structuring element with a diameter of 15, followed by a greyscale opening operation with the same structuring element. The purpose of the closing is to close holes that may appear in elements, and the purpose of the opening is to remove small, spurious, noisy elements

that may appear in the images. Finally, we applied a threshold, which is where the two models differ; the one optimised for time point 1 had its threshold set to 50, while the other one had it set to 30.

Figure 26 displays the application of the best model for a given time point to an arbitrarily chosen image from our validation set relative to our manual annotations. Obviously, we shall not draw any definitive



Time point 1 Time point 6 Time point 11

Figure 26: The best morphological model for 2D-projections for every time point evaluated against our annotations. Blue is true positive, black is true negative, red is false negative and green is false positive

conclusions based on an illustration on an arbitrarily chosen image from our validation set, but the figure is a good display of our struggles; the majority of objects in time point 1 are found, but there are still more false positives and negatives than we would have liked. Time points 6 and 11 are quite a lot worse; here, we consistently miss many large objects. See section 10 for the actual performance measures on the test set.

6.2 Morphology on the 3D-data

We also explored using a series of 3D-opening and closing operations on the 3D-stack after application of a derivative filter on each layer in the z-direction, as well as a 3D-derivative filter. The extra dimension of course means that the running time increases significantly, but nonetheless, we still explored this approach. As discussed in section 3.4, the z-axis in the 3D-data isn't a measure of horizontal position, but rather a series of images of the same location with different levels of focus. Due to this nature of the data, there is no basis for expecting a sharp change in values in the z-direction for elements in focus, as they'll typically slowly glide out of focus when moving along the z-dimension and keeping the x- and y-dimensions constant. Due to this, finding elements in focus with 3D-morphological operations approach seemed elusive.
We did run some experiments for this approach anyway, but as the results looked bleak, we decided to drop this approach altogether in favour of other, seemingly more promising approaches. Figure 27 displays one attempt on a smaller area, which illustrates why we stopped working with this approach.

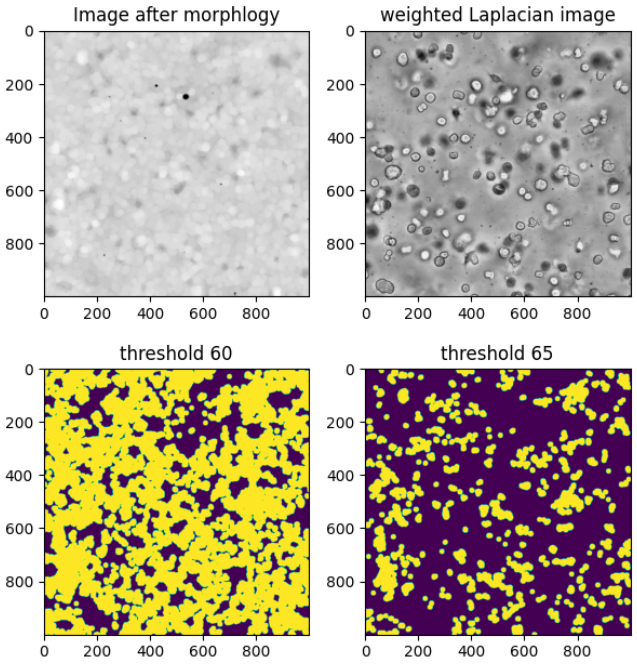


Figure 27: Top left: max projection after application of 3D-morphological operations. Top right: Weighted Laplacian projection of the same area. Bottom: 2 segmentations after application of thresholds and then max-projected to 2D. Yellow represents foreground and purple represents background

6.3 Layerwise morphology

We also built segmentation models where we apply a derivative filter to every layer in the z-direction, then process each of these layers individually with opening and closing operations, after which we stacked them back on top of each other, such that the data stays three dimensional, and we avoid the issues with three-dimensional morphology on our data as described in section 6.2. We have called this approach layerwise morphology, which we will refer to it as from now on. Do notice that this approach is functionally equivalent to the 3D-approach if we had only used structuring elements with an extent of 1 in the z-direction in the 3D-approach.

Just as in the 2D approach, we tried to optimise models for all time points as well as each of the three individual time points. This resulted in a total of 21 different sets of parameters that we evaluated on the validation set with this approach.
Here, it turned out that the best-performing model for time points 6 and 11 happened to be the same. Therefore, we have three models for this approach, one optimised for time point 1, one optimised for time points 6 and 11 and one optimised across all time points.
The optimal one for time points 6 and 11 is the application of a Sobel filter followed by four greyscale morphological operations: first closing with a circular structuring element with a radius of 5, then opening with a circular structuring element with a radius of 5, then closing with a circular structuring element with a radius of 30, then opening with a circular structuring element with a radius of 12. Finally, a threshold of 160 was applied. The initial small closing connects elements that are located near other structures and closes small holes. Following this up with a small opening removes many small, noisy elements that aren't connected to anything. This is followed up by a rather large closing, with the intention of closing large holes in objects, that typically appear as a consequence of derivative filters mostly finding edges. Then we finally apply another, larger opening to remove larger noisy elements and other non-focus objects.
This general approach of a small closing and small opening followed by a larger closing and opening has served us well for the layerwise morphological approach - so well in fact, that the two other layerwise morphological models, that performed well on time point 1 and across all time points, have the exact same morphological

operations: The time point 1 model is a Laplacian filter, followed by these morphological operations and a threshold of 100. The model across all time points is a Laplacian filter, followed by these morphological operations and a threshold of 60. This process is shown for the time point 1 model in Figure 28.

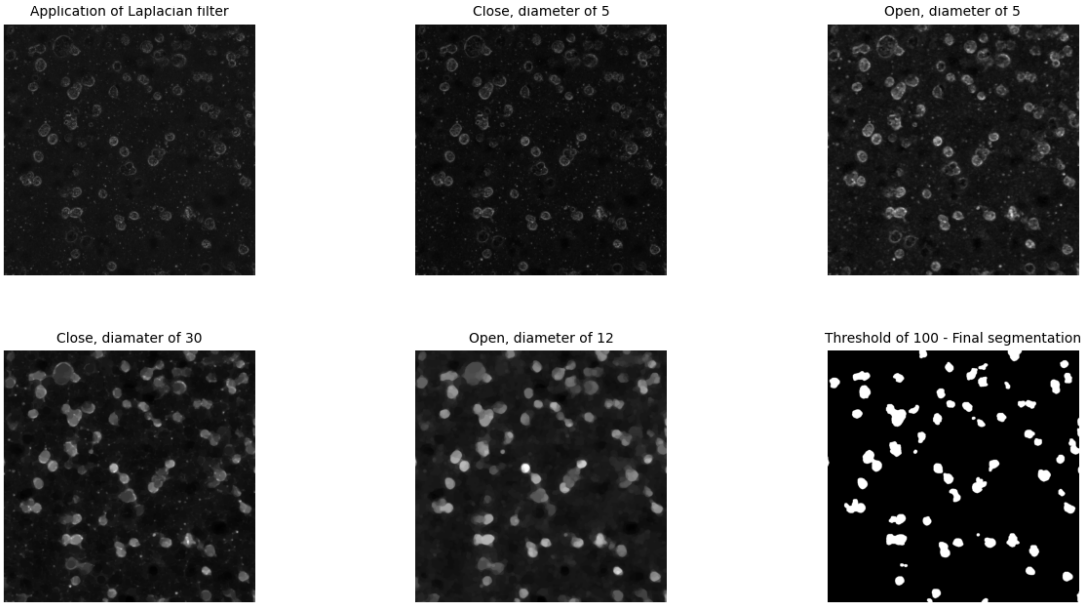
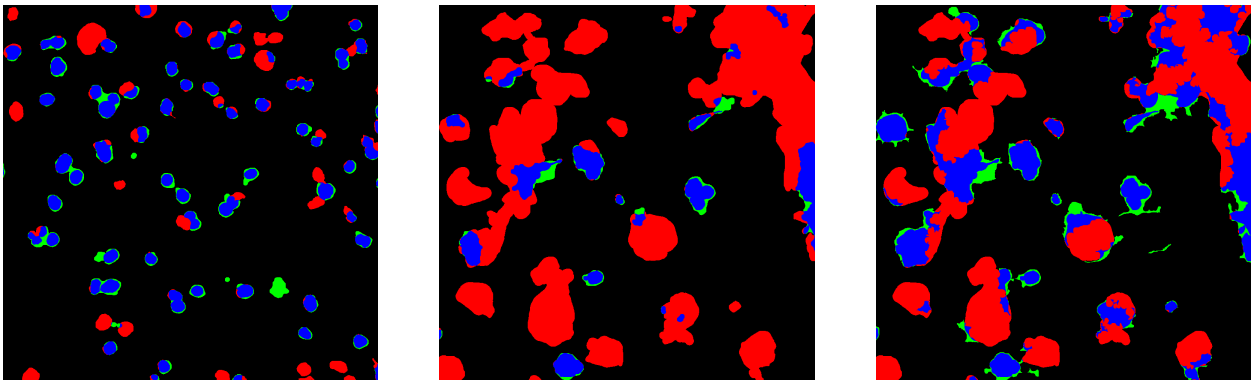


Figure 28: The morphological process for arriving at our segmentations applied to one image. The image is the same as otherwise used in this section, and the final segmentation is therefore identical to the one shown in 29. The six images in this figure are shown after the application of a max-projection for easier visualisation.

Figure 29 displays the application of the best model for a given time point to an arbitrarily chosen image from our validation set (the same one as used previously) relative to our manual annotations.



Time point 1 Time point 6 Time point 11

Figure 29: The best model for layerwise morphology for each time point evaluated against our annotations. Blue is true positive, black is true negative, red is false negative and green is false positive

Visually, the segmentations are quite similar to those from the 2D-morphological approach (see Figure 26) and this shouldn't come as a surprise; although one is based on a 2D-projection and the other isn't, they're both based on the same fundamental features, i.e., derivative values. Due to the similar nature of the segmentations, they also experience similar issues, especially in later time points. Again, these illustrations

are only for gaining a visual and intuitive idea of the performance of these models. See section 10 for the actual performance measures on the test set.

7 RootPainter

In the last decade, convolutional neural networks (CNNs) have been a well-used way of designing networks, when it comes to computer vision tasks, as in this case with image segmentation. CNNs are a layered model that learns different kernels for convolution, and have shown time and time again, as one of the reliable ways of constructing models for image analysis. A subclass of CNNs, such as the U-Net, has been shown to excel in segmentation tasks, which is the sort of problem we are trying to solve (Ronneberger, 2015). To make a model using a U-Net, one will need a set of images with corresponding annotations for the training, which we, in this case, are lacking, as there are no annotations for the images. This led us to a decision to make, either we make some annotations for images, or we use a tool like RootPainter, which uses a client–server architecture for making annotations while training a U-Net, and in collaboration, tell the model, which parts of its segmentation are correct (Ronneberger, 2015). Because we will have to annotate in both cases, and RootPainter can be trained to a high accuracy in different cases, with under 2 hours of annotation time according to their paper (Smith et al., 2022), we chose a path using RootPainter for training our U-Net models.

7.1 Setup of RootPainter

RootPainter comes in both a 2D- and a 3D-version, but because our available GPU is limited to 16 GB of usable RAM, we won't easily be able to use the 3D-version of RootPainter. When looking at other studies that have compared 2D- vs 3D-U-Net in segmentation tasks, there are small differences in their capabilities, but they both perform equally in their performance (Avesta et al., 2023). The dimensions of our data are also not the same, as the difference in the 3rd dimension is not placement but focus on the image. Because of this, we will use a 2D-U-Net trained using RootPainter to segment each layer in 2D and then combine into a 3D-segmentation.

RootPainter is an open-source project and can be downloaded from https://github.com/Abe404/root_painter. It consists of a server and a client, where the server can run on a different computer, but it must have access to a GPU with support for CUDA. The server handles the training of the model and the making of segmentations, during the annotation. The client handles the interaction with the user, where one can draw on the image, to tell which parts of the image are foreground and background. This will, in our case, be foreground for objects that are in focus, and background for everything else. The model's architecture is a variation of the U-Net, as it was designed by Ronneberger and is implemented in PyTorch. The U-Net is composed of a series of down- and up-blocks, which are joined by skip connections at each layer. The modifications to the U-Net have made it have an input size of 572x572, and an output size of 500x500, The input image is padded with 0s so the output segmentation matches the input image before padding. The number of trainable parameters has gone down from 31 million to 1.3 million. This will make it a rather small model, when compared to other leading image analysis models, but it has been shown to still be a reliable model with good performance (Smith et al., 2022). The model should learn to differentiate between foreground and background, given the user's annotations to the different parts of the images. During training, it uses some of the annotated images as a validation set. Then, for each epoch in the training, the pixel-wise F1 score is calculated, and the model is changed if it beats the previously best model. The training automatically stops if there have been no improvements for 60 epochs; this counter resets if a better model is found or more annotations are added. After training is complete, the model is ready to be used for the segmentation of the stacks. We will keep the default probability threshold of 0.5 for the model's prediction certainty for deciding when to predict positives.

7.2 Training of the models

We will be training 5 different models to use on the validation set, to see which is best for each time point and combined. One model with a dataset with images from all time points, one for each of the 1, 6, and 11 time

points separately, where the training and annotation happen simultaneously, and one where all annotations from the different models are put into one dataset, and a model is trained without this interaction with the model. To make our datasets, we take the 3D-stacks and make a crop out of a random patch from each layer, measuring 900x900 pixels. This is done from 6 randomly selected stacks, by random.org's generator, for each time point, making a total of 420 images for each time point, and 1260 combined for all time points. Then we annotate the images, which come in a random order, chosen by RootPainter's code. During the training of each model, we will be following the recommendations given by the creators of the RootPainter on how to perform the training. It can be found at: https://github.com/Abe404/root_painter/blob/master/docs/mini_guide.md.

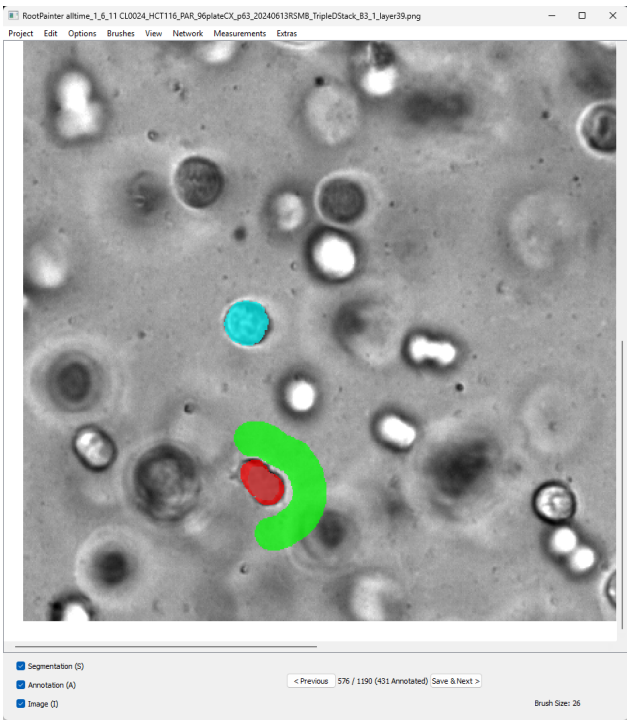


Figure 30: Annotation in RootPainter, for annotations: red is foreground and green is background. The segmentation from the model is cyan.

We will be annotating like this for approximately 2 hours per model, where we will let the model finish training. Figure 30 shows an image from the training process. During the annotation in RootPainter, we can get a plot of how well the current model performs and use it as an estimate of how much we will gain from further training. Figures 31-34 show the predicted-corrected area of the model's segmentations and the corrected annotations for images that the model has never seen. We see that the corrected-predicted area quickly decreases after the first 50 images, showing it has learned to segment quite well. Thereafter, small corrections keep coming, which don't seem to stop. Sometimes there comes a slight increase as the model has learned something unwanted, but it corrects itself afterwards. During this phase, we also learned that the model found many small elements, which didn't come from the desired objects. To remove these objects, we experimented with binary morphology and found that using an open operation using a ball with a radius of 10 was the best choice between removing unwanted objects and not removing wanted ones. This cleanup will be applied to all segmentations using models from RootPainter. An example of such cleaning can be seen in Figure 35.

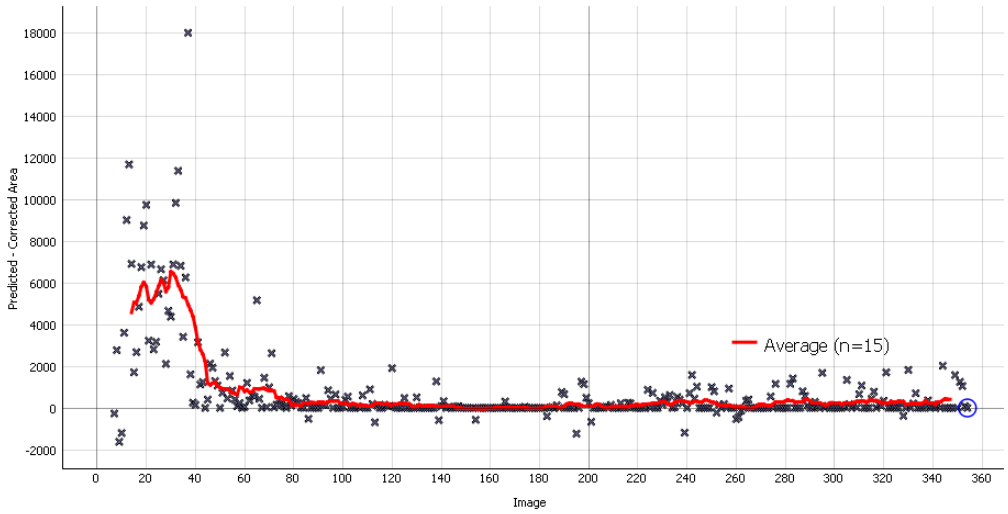


Figure 31: Predicted-corrected area from the training of the model on data from time point 1

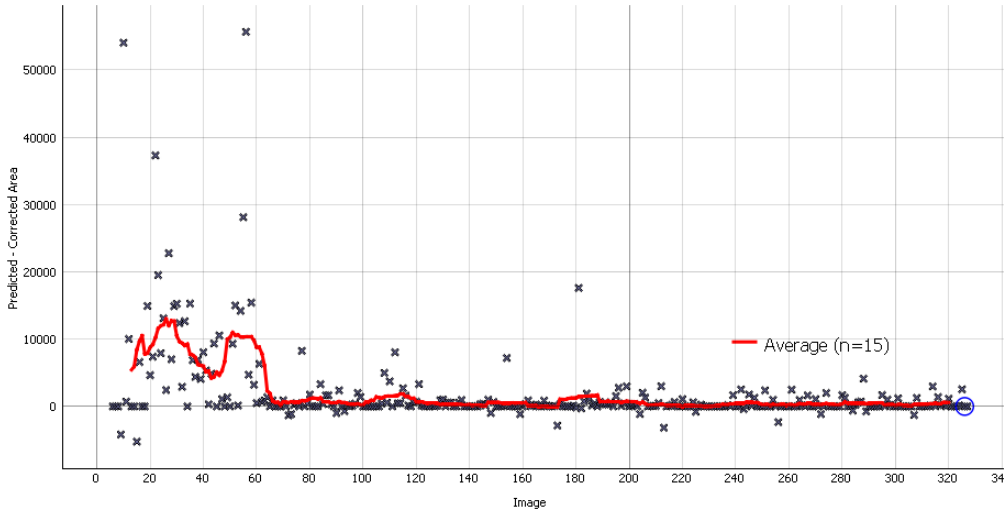


Figure 32: Predicted-corrected area from the training of the model on data from time point 6

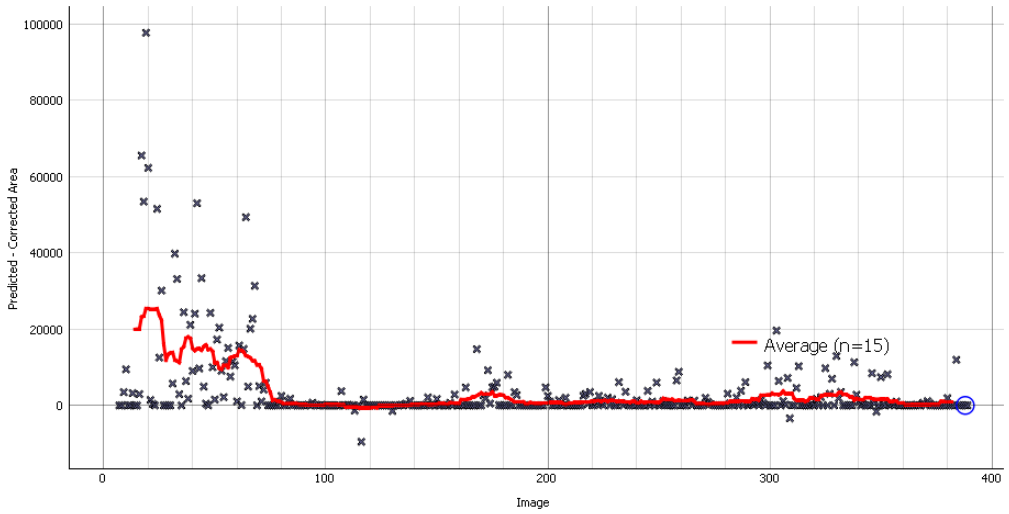


Figure 33: Predicted-corrected area from the training of the model on data from time point 11

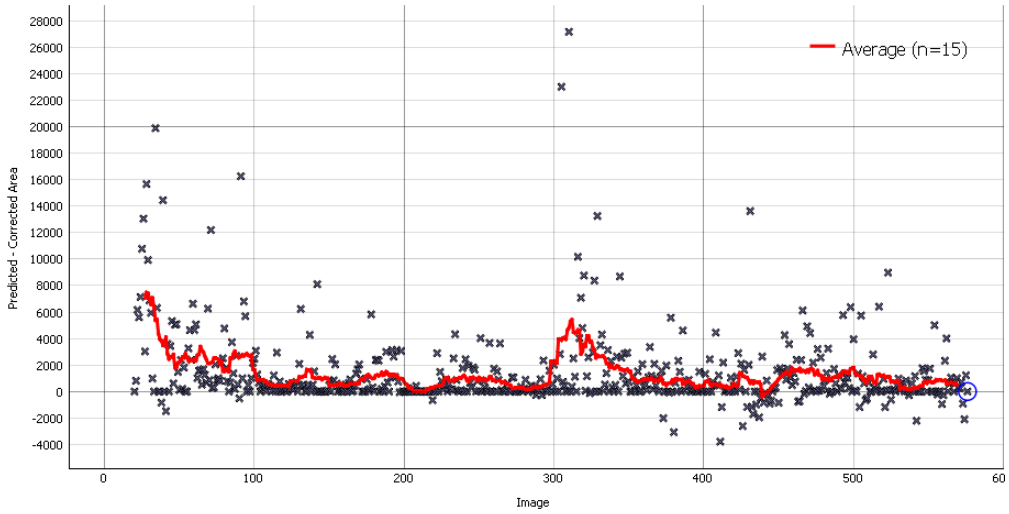


Figure 34: Predicted-corrected area from the training of the model on data from all time points

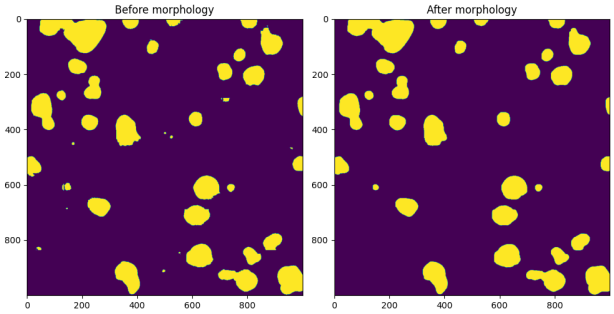


Figure 35: Result of using morphology to clean a 3D-segmentation. Yellow is foreground, purple is background.

7.3 Selection of models

After we created the 5 models, the models were then tested on our validation set, using the chosen metrics described in section 4, to find the ones that perform best for each time point and overall.

The best model for time point 1 is the model trained on all annotations from the 4 training runs, and for time points 6, 11, and combined, it is the model with a dataset of images from all time points. These two models will be used in our testing against the models from morphology and the company's model. From this, we saw that the 2 models trained on all time points were better than the ones trained on one specific time point, both for the evaluations run on one time point, and all time points combined. This can indicate that the training sets were not diverse enough to capture the differences the images can have at each time point, but were better captured by the general models with more diverse data from all time points. This could indicate that the models trained for singular time points might have been overfitted to their respective training sets.

8 Finding the layer of best focus

Part of the goal of this project is to find the z-layer in which a given object is in best focus. This can be seen as a different problem than xy-segmentation, as the z-axis has different properties. To do this, we will be using methods based on 2D- and 3D-segmentations of the stack. One key insight we hope to gain from this distinction is whether the company's decisions to project the stacks to 2D, such that it is compatible with YOLO, will harm their ability to subsequently find the objects' layer of best focus or not.

For the 2D-segmentation approach, we will analyse the original image stack using the xy-segmentations from our models. Our approaches for this will generally be using methods building on the paper (Pertuz, 2013), which describes how focus can be measured in an image, to find the layer in best focus. One of our approaches analyses the variance of the Laplacian values in each layer and sees how it changes; where the image is best in focus, is where the Laplacian variance is the highest. This is shown in Figure 36, where the cell in Figure 7 has been analysed and plotted. Here we can see a line which shows the Laplacian variance rises as the cell comes into focus and falls after. It also shows that the maximum variance is at layer 8, which also seems to be our depiction of the layer of best focus.

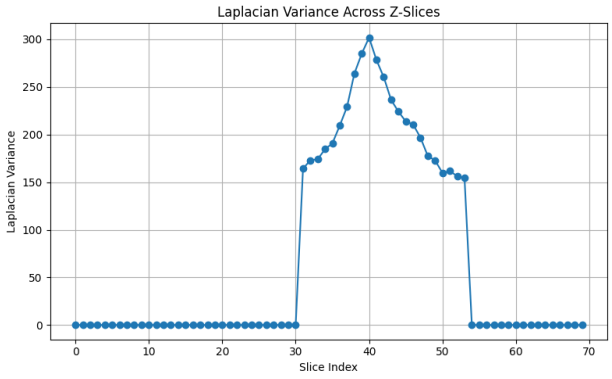


Figure 36: Plot of Laplacian variance for each layer for the cell in Figure 7

8.1 Using a 2D-segmentation

From a 2D-segmentation, we will find the bounding box that surrounds an object, and analyse each layer to then make a prediction on which is the best. We will examine the following methods of finding the layer of best focus:

The layer with the highest Laplacian variance

As described above, this indicates a method where the Laplacian variance is the highest is a way of finding the layer of best focus.

The layer with the highest Sobel magnitude variance

The argument for this method is that the Laplacian is a 2nd derivative filter, and the Sobel is a 1st derivative filter, so it may work with a lot of the same properties. We used Sobel kernels in the x and y direction to calculate the magnitude image of every layer and chose the one with the highest variance.

The layer with highest variance

Although quite naive, one could hope that the layer of best focus also has the highest variance, since whenever it is in focus, more details are visible. We are not very optimistic that this method will do well, because when we go out of focus, we get brighter or darker images, which can lead to higher or lower variances, but we try it to see if further image analysis can be left out, if a 2D-segmentation is used.

Average layer of highest Laplacian value for every xy-position

This is from the reasoning that, on average, the layer with the highest degree of focus has the highest Laplacian values. In this approach, we start by applying a Laplacian filter to all layers, then we choose the highest value in the z-axis from all layers for each xy-coordinate and make a 2D-map, where each pixel has the value of the corresponding layer. From the map, the average of the segmented area is used as an estimate of the layer of best focus.

8.2 Using a 3D-segmentation

To estimate the best layer from our 3D-segmentations, we will use the following two methods. They both build on the fact that our models tend to expand further in the z-axis as shown in Figure 37. Our approach then selects one layer from these, to decide which one is in best focus.

Middle layer

Using the 3D-segmentation, we will choose the middle layer where the object is represented. We expect our models to be just as likely to predict focus over and under the real layer, so the middle may be a good choice for the right layer. Although in cases where the layer of best focus is at the edge of the stack, we would expect this method to be poor.

Biggest layer

This method selects the largest layer of the 3D-segmented object. We suspect that the layer of best focus may be the largest segmented layer. The reasoning for this is that as objects move out of focus, not all of the area of the object is likely to be segmented.

8.3 Analysing z-layer estimation

When evaluating these methods, we will look at the error in 3 ways:

1: The mean and standard deviation of errors to see if the method tends to predict more over or under the target layer.

2: The mean and standard deviation of the absolute values of the error, to see how much it is off target on average.

3: The mean squared error (MSE), which will be our primary scoring method, as we want the scoring to penalise harder the further it is from the target. We will make one small adaptation to MSE, which involves reducing the error by 1 (or increasing) it by 1 if the error is negative) before squaring it. If the error is in the range $[-1, 1]$ it is set to 0 instead. We do this because the annotations are imperfect, and the layer next

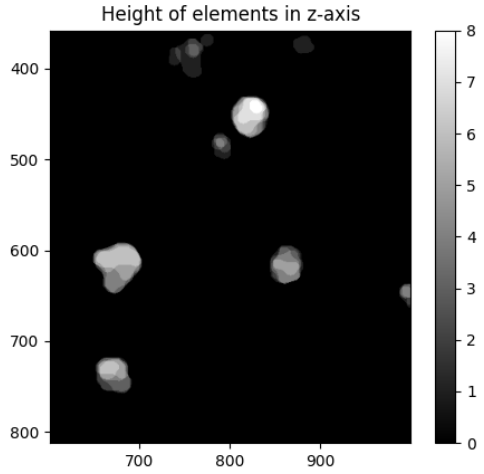


Figure 37: Height in z-axis of elements in a 3D-segmentation, using a RootPainter model

to the one we annotated as best will, in many cases, be just as well in focus. While making the annotations, we had to decide which layer was in best focus, and sometimes this was not possible to decide between two layers. And as we had to decide on one layer, we will allow one layer of error in the MSE calculation, because of this bias.

For the analysis and computing of metrics, we will use the 3D-segmentations of our models. To make sure that it is the right object we are comparing the segmentation to, we will first check if it has a 50% IoR overlap with an annotation, where the segmentation is considered the reference, before calculating the different predictions of the z-axis and comparing them to the layer of the given annotation. For the evaluation of methods using 2D-segmentations, we will use the segmentations from our RootPainter models to get a fair comparison between the RootPainter's 3D-segmentation and the 2D-methods. We will be analysing the results for each time point individually and for all time points.

9 Segmentation time

In this section, we will go through the segmentation time of the different models, as the time spent to segment each stack is a factor in the evaluation of the models. We have timed the segmentation on all stacks in the test set and will compare it to the company's time of approximately 40 minutes per stack as a yardstick. First, we will describe how we made the timings, then the results, and an analysis of this.

To get a fair comparison between the different models, all the times are gathered using one of our computers, except for the company's random forest, as we don't have access to the precise setup used by the company. The relevant specs of the computer used for the timings: CPU: i7-9700K, GPU: NVIDIA 2080 8 GB, RAM: 16 GB, where the computer used by the company had a more recent CPU and GPU, although we are not informed about the exact specs. Doing all the runs with timings, the system was never limited by RAM, and all other programs were closed, so as not to influence the compute time. All timings are without the time it takes to load the data from storage; the timing was started when the data was loaded in, and it was stopped when the final segmentation of one stack was produced. All times mentioned are averages. Since the 3D-stacks are sparse, the processing of the stacks is optimised to only run on the parts in 2D, where there is image data.

Time using RootPainter:

Since this method doesn't require any preprocessing, we timed our code from when we started to segment using the model to when it was finished, including the cleanup using morphology. The time it took to segment a stack in our test set on average was 70.103 s. The cleanup using morphology took on average 2.46 s., which makes the total time 72.503 s. per stack on average.

Time using morphology:

This method has two parts: creation of the 3D-Laplacian/Sobel stack or the weighted Sobel images, and the series of morphological operations, which have to be combined in the timing of the segmentation. The creation of a Laplacian stack is 3.638 s. on average, a Sobel stack is 5.361 s. on average, and the creation of the weighted Sobel image from the Sobel stack is 15.273 s. on average. For the morphology in 2D, a stack is processed in 0.827 s. on average, and in 3D it takes 35.396 s. on average. The combined times for 2D-morphology is 21.461 s., and for 3D it is 41.034 s., using a Laplacian stack, and 42.757 s., using a Sobel stack.

Our models all achieve at least a 33-fold improvement in segmentation speed per stack, successfully fulfilling our goal of accelerating the segmentation process compared to their approximately 40 min per stack. The morphology-based methods are the fastest overall, performing nearly twice as fast as the RootPainter models. Between the 2D- and 3D-morphology approaches, the 2D-method is only about twice as fast, due to the additional preprocessing required for 2D-segmentation.

10 Results

In this chapter, we will present the results from the testing of our different models with the measures described in section 4. First, we will go through the results from evaluating our segmentation for each time point and then the combined results across time points. Then we will present the results from estimating the layer of best focus, before analysing the results. In the following, a general model refers to a singular model that is trained for all time points and individually optimised for x means it is optimised for time point x. In Table 4, (individually optimised) means that it is aggregated from the individually optimised models for all time points.

First, some general sightings in our results: Sometimes, as in Table 1 for layerwise Morphology (general), we get a high recall but a low precision and therefore also a low F1 score because the model segments too much, and we therefore find many unwanted objects in the segmentation. In most cases, the scores based on pixel vs. objects give very similar results, but there were some instances where meaningful differences appeared. The biggest difference was in bad cases, as in Table 1 for layerwise Morphology (General), where the F1 value was more than cut in half when going from pixel to object analyses. If we had relied solely on pixel analysis, this model would not look as bad as it does when we look at it at object level.

Table 1: Overview of results on our test set for time point 1

Time point 1	Fractioned IoR F1 score ↑	Fractioned IoR Recall ↑	Fractioned IoR Precision ↑	IoR F1 score ↑	IoR Recall ↑	IoR Precision ↑	Pixelwise F1 score ↑	Pixelwise Recall ↑	Pixelwise Precision ↑
2D-Morphology (optimised for 1)	0.774	0.876	0.712	0.763	0.876	0.695	0.704	0.878	0.601
2D-Morphology (Optimised for 6) 2D-Morphology (Optimised for 11) 2D-Morphology (General)	0.242	0.995	0.140	0.232	0.995	0.133	0.466	**0.999**	0.305
layerwise Morphology (Optimised for 1)	**0.802**	0.812	0.825	**0.797**	0.812	0.815	0.727	0.777	0.723
layerwise Morphology (Optimised for 6) layerwise Morphology (Optimised for 11)	0.748	0.856	0.704	0.731	0.848	0.688	0.683	0.814	0.620
layerwise Morphology (General)	0.156	**1.000**	0.086	0.156	**1.000**	0.086	0.429	0.997	0.275
RootPainter (Optimised for 1)	0.692	0.561	**0.956**	0.691	0.561	**0.956**	0.703	0.602	**0.859**
RootPainter (Optimised for 6) RootPainter (Optimised for 11) RootPainter (General)	0.613	0.482	0.920	0.613	0.482	0.921	0.646	0.528	0.851
Random Forest	0.779	0.765	0.807	0.771	0.765	0.791	**0.739**	0.814	0.680

If we look at the results in Table 1, we see that the difference in the stacks from different time points shows in the results of the corresponding models. Because of this, models that are not optimised for the time point perform worse. All model approaches can get F1 scores around 0.7 or above, with the best being layerwise Morphology for time point 1, and it gets similar recall and precision values, which is a sign of good, balanced performance. This is despite it being a simple model when compared to the RootPainter and random forest models.

Table 2: Overview of results on our test set for time point 6

Time point 6	Fractioned IoR F1 score ↑	Fractioned IoR Recall ↑	Fractioned IoR Precision ↑	IoR F1 score ↑	IoR Recall ↑	IoR Precision ↑	Pixelwise F1 score ↑	Pixelwise Recall ↑	Pixelwise Precision ↑
2D-Morphology (Optimised for 1)	0.690	0.682	0.793	0.690	0.667	0.765	0.689	0.632	0.797
2D-Morphology (Optimised for 6) 2D-Morphology (Optimised for 11) 2D-Morphology (General)	0.554	**0.994**	0.398	0.512	**0.987**	0.356	0.697	0.949	0.562
layerwise Morphology (Optimised for 1)	0.584	0.527	0.902	0.563	0.513	0.889	0.614	0.519	0.869
layerwise Morphology (Optimised for 6) layerwise Morphology (Optimised for 11)	0.634	0.595	0.870	0.615	0.588	0.847	0.631	0.555	0.843
layerwise Morphology (General)	0.636	0.952	0.486	0.609	0.952	0.455	0.733	0.919	0.615
RootPainter (Optimised for 1)	0.770	0.683	**0.964**	0.770	0.683	**0.964**	0.773	0.707	**0.894**
RootPainter (Optimised for 6) RootPainter (Optimised for 11) RootPainter (General)	**0.800**	0.745	0.888	**0.800**	0.745	0.888	**0.792**	0.770	0.834
Random Forest	0.651	0.939	0.508	0.636	0.939	0.487	0.761	**0.952**	0.645

In this middle time point (Table 2), the difference between models optimised and not optimised for the given time point is not as big as it was for time point 1. This is expected as it lies between the two other time points, and therefore, the difference in cell growth from this time point to the others is the smallest. Only RootPainter can get over 0.7 in F1 score in all cases. Models like the random forest or Morphology get IoR recall scores around 0.95, so they find almost all objects, but find a lot of unwanted objects, which gives the lower precision and F1 scores. The RootPainter model has a better balance between recall and precision, meaning that more of the found objects are true objects.

Table 3: Overview of results on our test set for time point 11

Time point 11	Fractioned IoR F1 score ↑	Fractioned IoR Recall ↑	Fractioned IoR Precision ↑	IoR F1 score ↑	IoR Recall ↑	IoR Precision ↑	Pixelwise F1 score ↑	Pixelwise Recall ↑	Pixelwise Precision ↑
2D-Morphology (optimised for 1)	0.520	0.389	0.814	0.496	0.374	0.784	0.443	0.317	0.787
2D-Morphology (optimised for 6) 2D-Morphology (optimised for 11) 2D-Morphology (General)	0.608	0.765	0.510	0.553	0.708	0.463	0.641	0.674	0.619
layerwise Morphology (optimised for 1)	0.347	0.233	0.907	0.337	0.233	0.868	0.280	0.177	0.861
layerwise Morphology (optimised for 6) layerwise Morphology (optimised for 11)	0.330	0.218	0.867	0.320	0.218	0.808	0.289	0.189	0.843
layerwise Morphology (General)	0.640	0.678	0.650	0.605	0.653	0.602	0.636	0.638	0.662
RootPainter (optimised for 1)	0.548	0.440	**0.918**	0.539	0.440	**0.890**	0.581	0.492	**0.889**
RootPainter (optimised for 6) RootPainter (optimised for 11) RootPainter (General)	**0.698**	0.605	0.866	**0.682**	0.605	0.837	**0.738**	0.681	0.823
Random Forest	0.424	**0.969**	0.280	0.351	**0.969**	0.221	0.613	**0.951**	0.456

This (Table 3) was the hardest time point for the models as it has the lowest scores of the three time points. Only RootPainter achieved around 0.7 in F1 in all cases, where morphology was around 0.3-0.65. The layerwise Morphology (General) and the RootPainter model (General) had the best balance in recall and precision scores. The random forest model had a good recall of 0.969, but a very low precision, which indicates that too much was segmented as foreground.

Table 4: Overview of results on our test set across all time points

All time points	Fractioned IoR F1 score ↑	Fractioned IoR Recall ↑	Fractioned IoR Precision ↑	IoR F1 score ↑	IoR Recall ↑	IoR Precision ↑	Pixelwise F1 score ↑	Pixelwise Recall ↑	Pixelwise Precision ↑
2D-Morphology (General)	0.468	**0.917**	0.349	0.432	**0.896**	0.317	0.601	0.874	0.495
2D-Morphology (Individually optimised)	0.645	0.878	0.540	0.609	0.857	0.505	0.681	0.834	0.594
Layerwise Morphology (General)	0.477	0.877	0.407	0.457	0.868	0.381	0.600	0.851	0.518
Layerwise Morphology (Individually optimised)	0.589	0.542	0.854	0.577	0.539	0.823	0.549	0.507	0.803
RootPainter (General)	0.704	0.611	0.892	0.699	0.611	0.882	0.725	0.659	0.836
RootPainter (Individually optimised)	**0.730**	0.637	**0.903**	**0.725**	0.637	**0.894**	**0.744**	0.684	**0.839**
Random Forest	0.618	0.891	0.532	0.586	0.891	0.500	0.704	**0.906**	0.594

When comparing the results in Table 4, we can see that there is some performance to be obtained by making models for each time point. This can come at the cost of the models being more fitted to the time points and not being as robust as a model made for all time points would be. This is to be expected because the growth of the cells, whether they respond to the medicine or not, will be different, and the model must not be influenced by this, as it has to work in both cases. When comparing the results of the random forest and morphology models, we see that they perform similarly. Now, in regards to complexity, they both use Laplacian and Sobel filters, but the random forest uses them on different scales, and it uses other more complex techniques such as structure tensors, making it a more complex and time-consuming model, but in our results, it doesn't improve the performance. This could point to errors in the random forest model, but it can also come from bad annotations, overfitting, it being a bad model for the problem or something else. The morphology models and the random forest have better recall values compared to RootPainter models, meaning that those models find more of the desired objects. The models from RootPainter have better precision values across all time points, meaning they have more correctly segmented objects in their segmentation. The RootPainter usually has a lower recall than precision score, and it could maybe be balanced more if we had chosen a lower certainty threshold for a pixel being in focus in the model.

Table 5: Overview of results of methods for estimating the layer of best focus from a 3D-segmentation. MSE is calculated according to the description in section 8.

3D-segmentation	Time point	Method	MSE (↓)	Abs. dist. ± std. (↓)	Dist. ± std. (→ 0)
RootPainter	1	Middle layer	0.291	0.838±0.736	-0.562±0.962
		Biggest layer	0.363	0.900±0.846	-0.675±1.034
	6	Middle layer	1.436	1.365±1.217	-0.865±1.611
		Biggest layer	2.176	1.568±1.453	-1.054±1.859
	11	Middle layer	2.242	1.534±1.452	-0.856±1.931
		Biggest layer	3.017	1.525±1.750	-0.746±2.199
	All	Middle layer	1.440	1.266±1.258	-0.818±1.586
		Biggest layer	1.975	1.320±1.486	-0.808±1.816
Morphology	1	Middle layer	3.911	1.276±2.017	0.047±2.386
		Biggest layer	4.827	1.795±2.101	-0.850±2.630
	6	Middle layer	11.087	2.900±2.761	1.110±3.840
		Biggest layer	10.844	2.697±2.856	0.697±3.866
	11	Middle layer	11.733	2.541±3.084	0.116±3.994
		Biggest layer	11.589	2.411±3.131	0.137±3.949
	All	Middle layer	38.714	4.859±4.888	3.304±6.049
		Biggest layer	38.880	4.731±5.009	2.343±4.780

Table 6: Overview of results of various methods for estimating the layer of best focus from a 2D-segmentation. MSE is calculated according to the description in section 8.

Time point	Method	MSE (↓)	Abs. dist. ± std. (↓)	Dist. ± std. (→ 0)
1	Highest Laplacian variance	9.604	1.302±3.144	0.279±3.3913
	Highest Sobel variance	18.930	2.813±3.984	1.976±4.459
	Highest variance	98.65	8.763±5.898	5.348±9.009
	Highest Laplacian layer	75.84	8.638±4.181	-2.868± 9.159
6	Highest Laplacian variance	10.404	1.829±3.171	−0.255±3.652
	Highest Sobel variance	19.595	2.893±4.027	1.319±4.780
	Highest variance	129.276	9.702±7.322	7.659±9.437
	Highest Laplacian layer	59.462	8.078±4.293	-1.508±8.708
11	Highest Laplacian variance	13.131	2.000±3.524	-1.894±3.582
	Highest Sobel variance	7.421	1.842±2.650	0.052±3.227
	Highest variance	128.631	9.973±6.937	4.447±11.306
	Highest Laplacian layer	64.955	7.760±4.389	-3.508±8.196
All	Highest Laplacian variance	10.945	1.703±3.284	-0.562±3.656
	Highest Sobel variance	15.757	2.554±3.686	1.164±4.331
	Highest variance	118.796	9.460±6.874	5.929±10.080
	Highest Laplacian layer	66.595	7.938±4.298	-2.559±8.657

To analyse the different methods of obtaining the layer of best focus, we will examine the results from Table 5 and 6. The best methods for estimating the z-layer in best focus are based on the 3D-segmentaions from RootPainter.

If we look at the results using the 3D-segmentation in Table 5, we see a pattern that the layer is harder to estimate in later time points, as the MSE increases. If we examine the mean distance for RootPainter models, they predict a layer that is too low on average, with an absolute mean error of up to around 1.5. The standard deviation tops around 2, meaning that the results are fairly consistent. This results in the lowest MSE scores, so it was the best model for this purpose.

When looking at the results from using the Morphology 3D-segmentations, the same increase in MSE in later time points is also seen. The general model for all time points is performing much worse than the individual ones, which shows that one model with Morphology can't be used if the z-axis is desired. The mean errors are generally very comparable to those from the RootPainter, but the standard deviation is bigger, so the models are not so consistent in their predictions, and it has bigger errors shown by the absolute errors, as well as a worse score in MSE.

If we compare the two methods (biggest layer and middle layer) of finding the layer from 3D-segmentations it is not clear whether one is better than the other, as they score nearly identically, although the middle layer method does seem to have a slight edge.

When looking at the scores from the different methods using a 2D-segmentation, shown in Table 6, to estimate the z-axis using the original stack, we see that the method using the layer of highest Laplacian variance worked the best. It had scores similar to those of 3D-segmentation using Morphology, which was worse than 3D-segmentations from RootPainter. The method using the Sobel Variance also seems like a working method; it did manage to beat the Laplacian variance at time point 11, but was overall worse. The two other methods in 2D didn't perform well, as their estimations were over 7 layers off on average and with high standard deviations.

11 Discussion

We will now discuss various aspects of our work and highlight key insights as well as other meaningful implications. This includes biases, uncertainty, implications of how our models could be applied, our ability to find the layer of best focus and more.

11.1 Biases

The results should, of course, be viewed in the light of the biases present in the project. One bias appears as a consequence of the production of annotations discussed in section 5. Since we annotated the test set after we finished training our models, we were unable to use our knowledge of the test set in the training of the models, but this does conversely mean we knew about our models' strengths and weaknesses when annotating. We did, of course, do our utmost not letting ourselves be influenced by this knowledge of our models when annotating, but it is impossible to ensure that we weren't subconsciously affected to some degree.
Another bias, which is important to notice, is selection bias, which is present as a consequence of testing multiple models on the test set and picking the best one. In this case, the selection bias takes the form of data leakage, as the best-performing model is picked out in relation to the other models, whereby the test set has impacted the model choice, and therefore it can not be considered entirely unbiased.
We were aware of this when running the tests, but as we are primarily interested in the comparative analysis among models, we are less interested in the exact scores of the best model and more so in looking for an indication of what sort of approach appears to have the ability to perform well. Consequently, if one wanted an estimation of model performance of the best-performing model, it would be necessary to gather an entirely new and annotated set of data that has not been used in training, validation or testing.

11.2 Robustness of models

As discussed in section 5, our test set regarding the xy-plane consisted of 12 cropouts of 1000x1000 pixels each, 4 cropouts for each time point. This is smaller than what would be ideal, and therefore, this raises questions about how much our results could have been swayed by variability. In order to inspect this, we calculated the standard deviation and the 95%-confidence interval of fractioned F1 scores on the test set above for the RootPainter (individually optimised) and the random forest. For the RootPainter, the resulting standard deviation is 0.130 and the 95%-confidence interval is $[0.657, 0.804]$, while for the random forest the standard deviation is 0.194 and the 95%-confidence interval is $[0.508, 0.728]$. This does imply that the RootPainter appears to be less variable, which is a sign of it being more robust than the random forest. The confidence intervals are, unsurprisingly, quite large, and the intervals for the RootPainter and the random forest overlap, which is another reason why all conclusions in this regard should be taken with this large uncertainty in mind.
We have, for simplicity and clarity, not included these calculations for other models, but the other standard deviations and confidence intervals are similar.

11.3 The ability to generalise to new data

Most of the models face different difficulties when presented with new data that doesn't resemble the training set. The Morphology model may be mostly invariant to the texture of the cells, but it seems very sensitive to the lighting level in the images, which can change due to different growth of the cells, as described in section 3. This is evident, as the biggest difference between the model's setups is the thresholds used, which yield very different results. Here, layerwise Morphology Optimised for time point 1 uses a threshold of 100, and layerwise Morphology general uses 60, while both use a Laplacian filter on the stack. From our data, we see that a lower threshold is better for late time points, and a higher one is better for earlier time points.

For RootPainter, it needs to have these variants in its dataset before it can be used on new, different data. Because if the cells look different or the lighting changes, it will change how the cells look, which can negatively impact the performance. In both cases, this new data can be added to the existing model, and with a little more annotation, a new model can be made after retraining the model.
So if both models are to be used on new data that looks different, it will have to be accounted for and will therefore be a factor that has to be monitored continuously.

11.4 The value of designing models for every time point

As can be seen in Table 4 and as stated in the accompanying discussion, individually optimised models tend to score better than their general counterparts. This is especially evident for the morphological approaches, where the difference in F1 values for individually optimised and general models is larger, and this could be an implication that designing models for individual time points could improve performance. This does not come as a surprise when one considers the simplicity of the morphological approaches.
The RootPainter, on the other hand, expresses a very small difference, and it is therefore difficult to say whether this gain is a consequence of the large variance of the test set. Having a general model also has some advantages, as one would expect it to be more robust, which might especially come in handy when observing the effects of cells growing in various ways after application of medications.

11.5 Use of the segmentations

If the segmentations are to be used further down the company's data pipeline, where the objects are classified and analysed, every model either finds extra unwanted objects with higher recall but lower precision, fewer correct objects, but with a higher precision score or both of these to some degree.
Whether the company prefers higher recall or precision, we don't know, but it is a choice they will have to make if they use one of these models. As the company intended the segmentations to be used for training a YOLO model, the performance of the resulting YOLO model can be hurt from errors showing in low recall and/or precision. Since YOLO learn from the annotations given to it, its ability to learn and generalise would be hurt if the annotations contain many errors (Koksal et al., 2020). For this case, an argument can be made that, when generating segmentations for YOLO the impact of low recall and precision perhaps don't impact the training equally, therefore the results may have to be reanalyzed with a F_β score where β is a factor of the importance of recall over precision (Metrics Reloaded, 2025a). If higher recall is preferred, the best choice can change to be their random forest, but this will have to be examined more deeply before a choice can be made. Another aspect is the performance difference across time points. The results show their random forest model and most of the other models keep falling in performance when going from time point 1 to 11. This can lead to a model with better performance in the early time points, but it is in later time points that they can see if the cells have grown, which could prove to be an issue. From this view, the models from RootPainter have a more even score across the time points.

11.6 Can a layer of focus be found

Our results show that to get the best estimate of the layer of best focus, one will need to make a good segmentation in 3D, and a fairly good estimate can be found from a 2D-segmentation. There may be other ways to calculate it than the ones tried, but according to our results, a good 3D-segmentation is the best approach. If the company wants to use YOLO as their model, they would benefit from doing it the same way as we did, where each layer is segmented to get a 3D-segmentation, as it was the most accurate both in terms of absolute average error, MSE, and it has the lowest standard deviations of the estimates.

11.7 Combined evaluation of models

All in all, the individually optimised RootPainter model is the one providing segmentations of the highest quality according to our methods for evaluating this. It takes about twice as long as the morphological approach, but we would argue that this is of lesser importance for multiple reasons. First, both RootPainter and morphological approaches are significantly faster than the method established by the RootPainter, which was the goal of the time aspect of the models. Second, if the segmentations were to be used for the training of a YOLO model, we would only need to segment the stacks once, as the YOLO model would be the one being used subsequently. And last, the quality of segmentation has throughout the entire project taken priority over the time aspect.

11.8 Is YOLO a good way forward

If we look at all our findings, could YOLO be a good choice of model? As YOLO is a machine learning model with some of the same building blocks and structure as RootPainter(Ultralytics, 2025), it should be able to get comparable results with our models from RootPainter, which was the best-performing.
If a YOLO model is given the 2D-projections as the company currently does, it will not be able to provide the third axis as reliably as from a 3D-segmentation. However, if they use the model in a similar way to how we used RootPainter, segmenting 2D-images to create a combined 3D-segmentation, it should be possible to get it as good as we did. It would require some annotations of good quality, as the performance will fall if they are subpar (Koksal et al., 2020). We lack high amounts of quality segmentations, and this would be an argument for using RootPainter, but as the annotations only have to be made once, it could depend on whether there is a speed or performance increase when using YOLO, and if it is worth the extra work.

12 Conclusion

In this project, we wanted to examine different ways of finding objects in focus in the image stacks from the company without using YOLO. In that we have succeeded as we have found two reasonably reliable ways: one with image filters and morphology, and one with RootPainter, which both demonstrate comparable or in some cases even better performance than the company's random forest model from Ilastik. The model from RootPainter was the best performing of them all, and it would be our recommendation to use it in further work.
Another part of this project was to decrease the computational time needed, which we also succeeded in with a minimum of 33-fold improvement for our models per stack.
To answer whether a 3D-segmentation is needed to find the third axis in the stack, our results point to it being the best way, but a good estimate can also be found from a 2D-segmentation.

13 Future work

We see multiple avenues that one could explore for future work.
One such option is exploring how the performance of relevant YOLO models is affected by the precision-recall tradeoff. As mentioned, past research indicates that these values aren't of equal performance, and one might look into their individual relevance in relation to the YOLO architectures the company is interested in training. Doing this would enrich our knowledge of how to optimally evaluate segmentations, which shall be used for the training of a YOLO model.
Another exciting thing to explore could be how much the performance of the RootPainter could be improved. We see some general ways one could approach this. First, investing more time in the training and annotation process is something we would imagine could improve its performance. Second, gathering a larger dataset, either through more lab work or other methods such as data augmentation, could also improve its performance. A third thing could be to change some of the hyperparameters in RootPainter, e.g. the probability of predicting positively, which was left to the default value of 0.5, but the models had lower recall scores than precision, so there could be a better choice with a better balance between the scores. And last, looking into alternate ways of handling the data at the preprocessing stage, such as scale selection, might also lead to approaches that could improve performance.

14 Reference list

Avesta, A., Hossain, S., Lin, M., Aboian, M., Krumholz, H. M., & Aneja, S. (2023). Comparing 3d, 2.5d, and 2d approaches to brain image auto-segmentation. *Bioengineering*, *10*(2). https://doi.org/10.3390/bioengineering10020181

Fredborg, M., Andersen, K., Spillum, E., Droce, A., Olesen, T., Jensen, B., Rosenvinge, F., & Sondergaard, T. (2013). Real-time optical antimicrobial susceptibility testing. *Journal of clinical microbiology*, *51*. https://doi.org/10.1128/JCM.00440-13

Ilastik. (2025). Ilastik - the interactive learning and segmentation toolkit [Accessed: 28.05.2025]. https://www.ilastik.org/

Koksal, A., Ince, K. G., & Alatan, A. A. (2020). Effect of annotation errors on drone detection with yolov3. *CoRR*, *abs/2004.01059*. https://arxiv.org/abs/2004.01059

Metrics Reloaded. (2025a). F beta score [Accessed: 21.05.2025]. https://metrics-reloaded.dkfz.de/metric?id=f_beta

Metrics Reloaded. (2025b). Intersection over reference (ior) [Accessed: 21.05.2025]. https://metrics-reloaded.dkfz.de/metric?id=intersection_over_reference

Pertuz, O. (2013). Analysis of focus measure operators for shape-from-focus. *Pattern Recognition*, *46*, 1415–1432. https://doi.org/https://doi.org/10.1016/j.patcog.2012.11.011

Random.org. (2025). Random.org [Accessed: 23.04.2025]. https://www.random.org

Ronneberger, O. (2015). U-net: Convolutional networks for biomedical image segmentation. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, *9351*, 891–921. https://doi.org/https://doi.org/10.1007/978-3-319-24574-4_28

Smith, A. G., Han, E., Petersen, J., Olsen, N. A. F., Giese, C., Athmann, M., Dresbøll, D. B., & Thorup-Kristensen, K. (2022). Rootpainter: Deep learning segmentation of biological images with corrective annotation. *New Phytologist*, *236*(2), 774–791. https://doi.org/https://doi.org/10.1111/nph.18387

Ultralytics. (2025). Yolo11 [Accessed: 28.05.2025]. https://docs.ultralytics.com/models/yolo11/

15 Appendix

15.1 Full 2D-projections (downscaled to decrease image sizes)

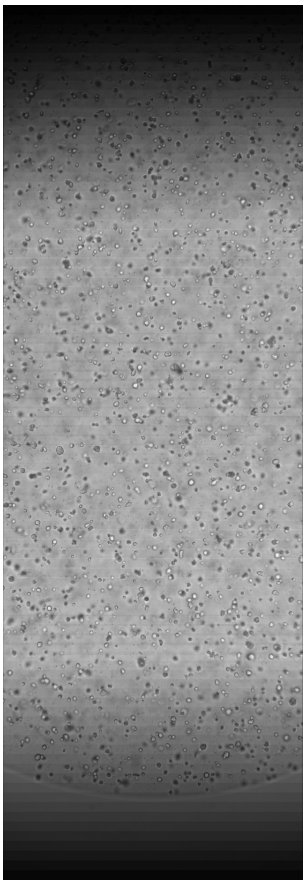


Figure 38: Mean projection

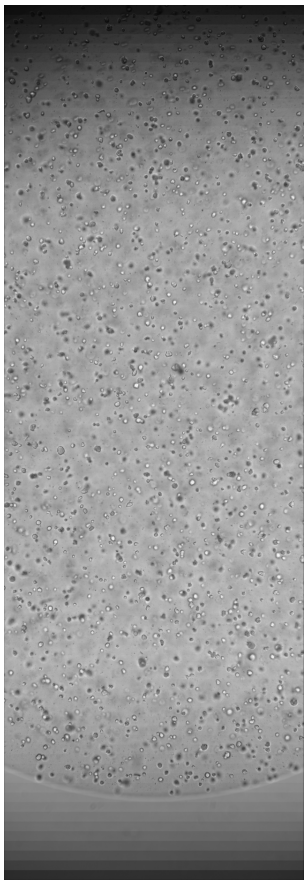


Figure 39: Standard deviation projection

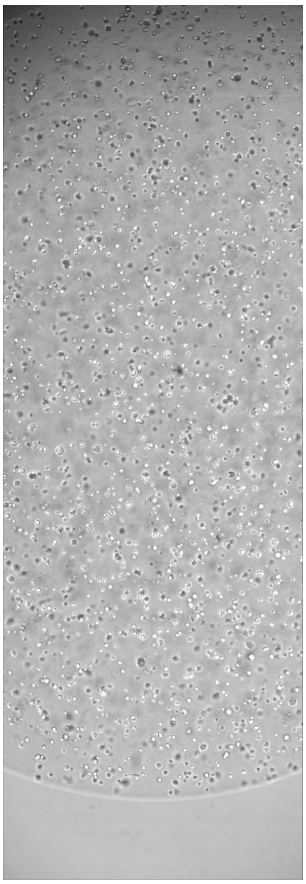


Figure 40: Max projection

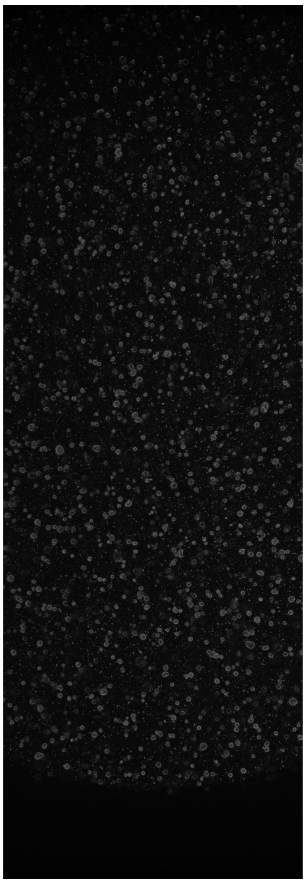


Figure 41: Max intensity projection after application of a Sobel filter

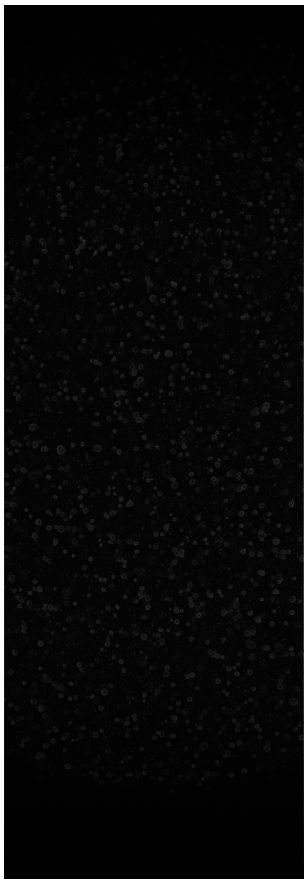


Figure 42: Mean intensity projection after application of a Sobel filter

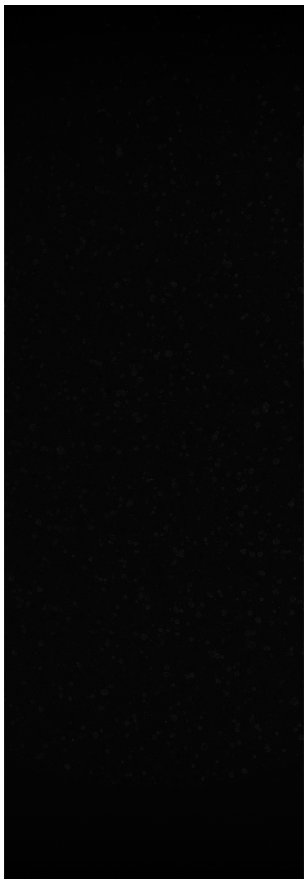


Figure 43: Max intensity projection after application of a Laplacian filter

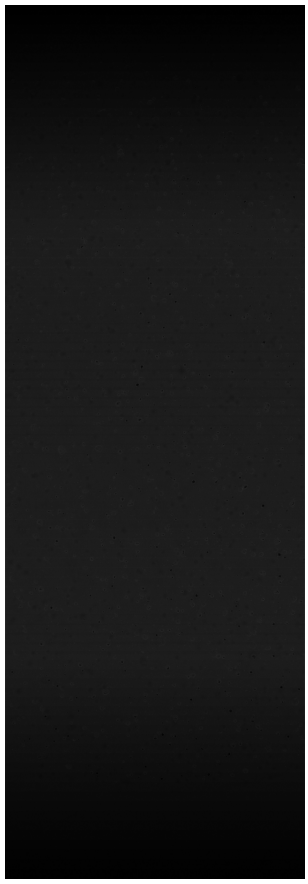


Figure 44: Mean intensity projection after application of a Laplacian filter

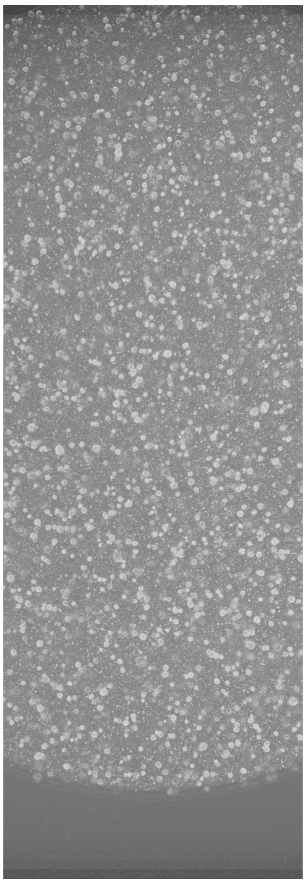


Figure 45: Max intensity projection after application of a Sobel filter on a logarithmic scale

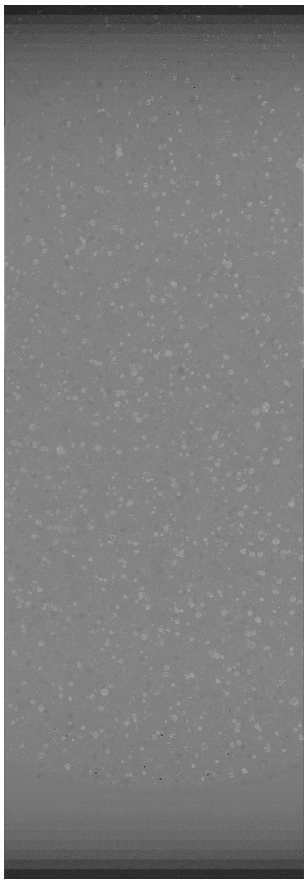


Figure 46: Max intensity projection after application of a Laplacian filter on a logarithmic scale

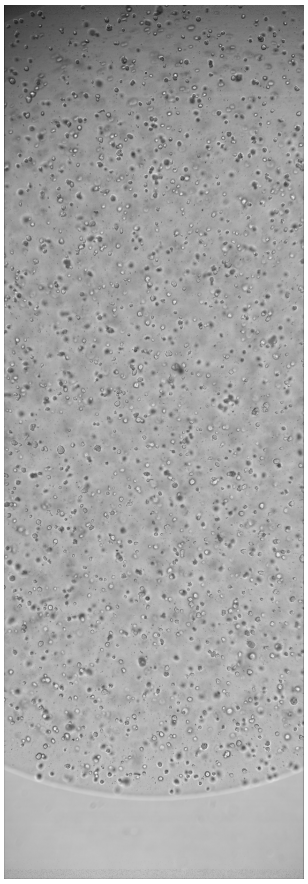


Figure 47: Application of a Laplacian filter followed by a weighted sum of the original image based on Laplacian magnitude

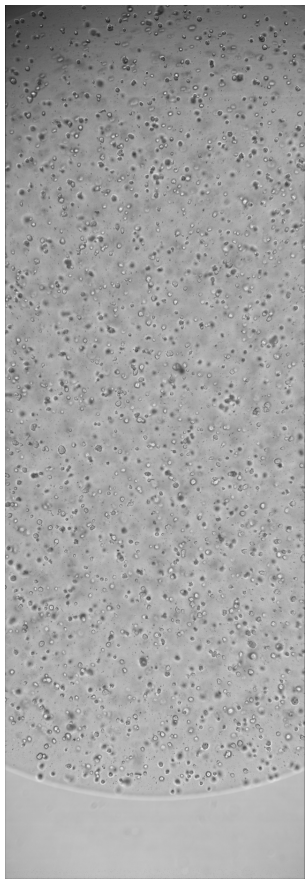


Figure 48: Application of a Sobel filter followed by a weighted sum of the original image based on Sobel magnitude

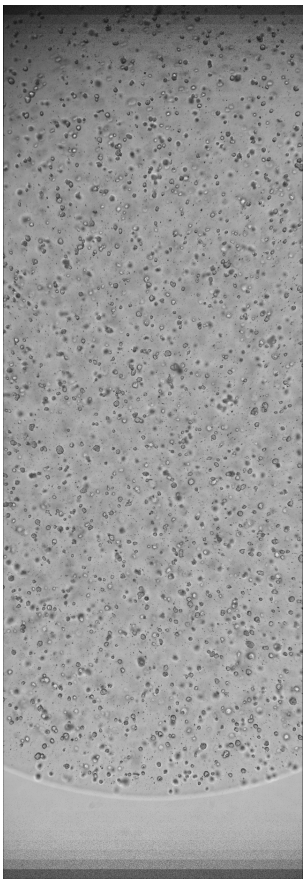


Figure 49: Application of a Laplacian filter, followed by taking the original value of the layer with the highest Laplacian value

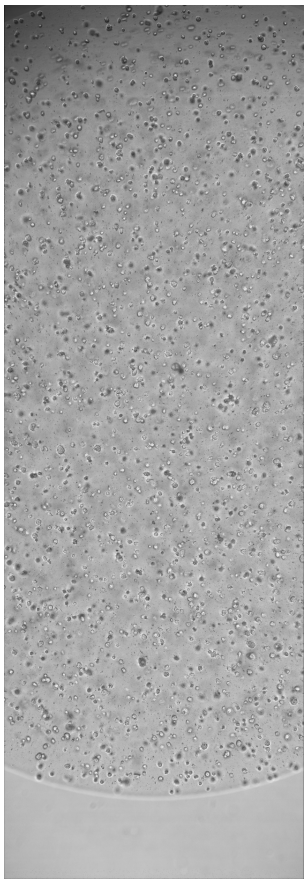


Figure 50: Application of a Sobel filter, followed by taking the original value of the layer with the highest Sobel magnitude

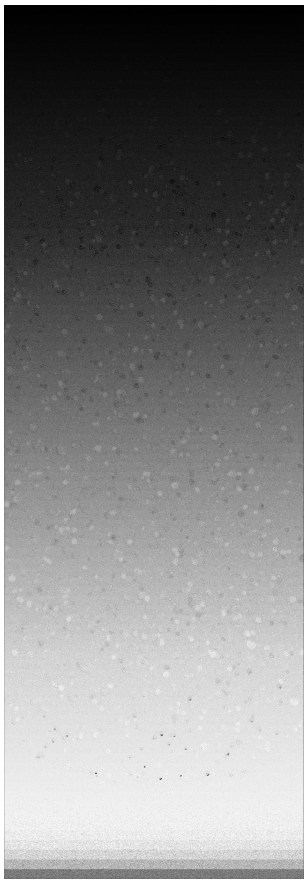


Figure 51: The layer with the highest Laplacian value

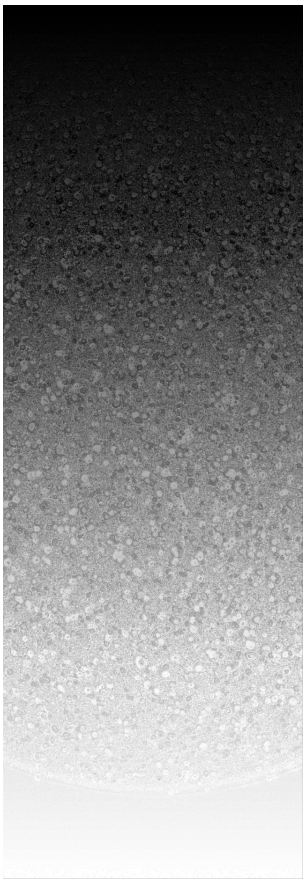


Figure 52: The layer with the highest Sobel magnitude

15.2 AI Deklaration

Deklaration for anvendelse af generative AI-værktøjer (studerende)

På Københavns Universitet udfører vi vores arbejde med **ansvarsfølelse** og **respekt for samfund, kulturarv, miljø og mennesker** omkring os.
Integritet, ærlighed og **transparens** er forudsætninger for akademisk arbejde. Vi forventer derfor, at eksamenspræstationer afspejler **den studerendes egen læring og selvstændige indsats**.
Akademisk arbejde baserer sig altid på andres indsigter, viden og bidrag, men altid med **grundig anerkendelse, respekt** og **kreditering** af dette arbejde.
Dette gælder også ved brug af generativ kunstig intelligens.

Vejledning

Brug af generativ AI ved eksamen

I henhold til KU's regler for brugen af værktøjer, der er baseret på generativ AI (GAI), skal du være transparent om din anvendelse af teknologien, fx i dit metodeafsnit og/eller ved at udfylde og vedlægge nedenstående deklarationsskabelon som bilag til skriftlige opgavebesvarelser.
Når du skriver din deklaration, er det vigtigt, at læseren får et tydeligt billede af, om og hvordan generativ AI har bidraget til det endelige produkt.

Hvis det er besluttet, at du skal bruge skabelonen i dit fag, skal du også benytte den, når du *ikke* har anvendt GAI-værktøjer som hjælpemiddel. I dette tilfælde skal du dog blot krydse af, at du ikke har brugt GAI, og behøver ikke at udfylde resten.
Ved at deklarere din brug af GAI-værktøjer sikrer du, at der ikke opstår udfordringer i forhold til reglerne om eksamenssnyd.

I kurser, hvor brugen af GAI er integreret i fagligheden, kan refleksion og kritisk vurdering af anvendelsen af GAI-værktøjer også indgå som en del af et metodeafsnit i din opgave. Spørg din underviser eller vejleder, hvis du er tvivl, om det er tilfældet i dit kursus.

Hvis generativ AI er objekt for din undersøgelse, vil det fremgå af dine forskningsspørgsmål, din metodebeskrivelse, din analyse og konklusion, hvilken rolle GAI spiller i din opgave. Hvis du

samtidig også bruger GAI som hjælpemiddel i processen, skal du deklarere denne anvendelse særskilt.

Opmærksomhedspunkter:

- Hvis GAI er et tilladt hjælpemiddel på dit kursus, må du anvende GAI til dialog og sparring under udarbejdelsen af din opgave, men du må **ikke** overlade udfærdigelsen af din opgavebesvarelse til GAI-værktøjer, selvom alle hjælpemidler er tilladt.
- Hvis materiale fra GAI inkluderes som kilde (direkte eller i redigeret form) i din besvarelse, gælder de samme krav om brug af citationstegn og kildehenvisning som ved alle andre kilder, da der ellers vil være tale om plagiat.
- Brug aldrig personhenførbare, ophavsretsbeskyttede eller fortrolige data i et AI-værktøj.
- Husk altid at undersøge gældende regler og retningslinjer for brug af generativ AI på KU.
- Læs kursusbeskrivelsen grundigt. Det er vigtigt at du ved, hvilke anvendelser der er tilladt i dit kursus. Der kan eventuelt være yderligere krav om dokumentation, fx at du skal beskrive dine centrale prompts og evt. kildemateriale (hvad har du givet af kontekst, hvad har du fodret værktøjet med, hvad har du bedt værktøjet om at gøre), beskrive outputtet (hvilke svar du fik af værktøjet), beskrive processen, f.eks. historik og iterationer (hvis du har skrevet frem og tilbage med værktøjet ad flere omgange for at komme frem til et brugbart svar).
- Tal med din underviser eller vejleder, hvis du er i tvivl.

Deklaration for anvendelse af generative AI-værktøjer (studerende)

☒ **Jeg/vi har benyttet generativ AI som hjælpemiddel/værktøj** *(sæt kryds)*

☐ **Jeg/vi har <u>IKKE</u> benyttet generativ AI som hjælpemiddel/værktøj** *(sæt kryds)*

Hvis brug af generativ AI er tilladt til eksamen, men du ikke har benyttet det i din opgave, skal du blot krydse af, at du ikke har brugt GAI, og behøver ikke at udfylde resten.

Oplist, hvilke GAI-værktøjer der er benyttet, inkl. link til platformen (hvis muligt):

ChatGPT, https://chatgpt.com/

Beskriv hvordan generativ AI er anvendt i opgaven:

1) *Formål (hvad har du/I brugt værktøjet til)*
   Vi har benyttet GAI til at generere struktur og forslag til brug af python biblioteker i koden.
2) *Arbejdsfase (hvornår i arbejdsprocessen har du/I brugt GAI)*
   Igennem hele arbejdsprocessen.
3) *Hvad gjorde du/I med outputtet (herunder også, om du/I har redigeret outputtet og arbejdet videre med det)*
   Vi bearbejdede og tilpasset outputtet sådan, at det passede specifikt til det vi skulle bruge det til.

NB. GAI-genereret indhold brugt som kilde i opgaven kræver korrekt brug af citationstegn og kildehenvisning. Læs retningslinjer fra Københavns Universitetsbibliotek på KUnet.

15.3 How to run the code

Rootpainter

To run the Rootpainter models, one will first have to follow the guide from its repository to set up a Python environment to run the code. to run the code on the test stacks: 1) in use_on_stacks_test_all.py there is a function run_model_on_testset() that needs the path of the original stacks.
2) in crop_stack_to_test.py use crop_results() with set the paths to make the right cropouts 3) if morphology cleaning is wanted use the function in morph_clean.py to clean all files in a folder.

Evaluate z-errors

Use the file Evaluation/z-analyser.py and change the paths on lines 143-145 to the correct paths. Does only takes 3D-segmentations.

projection of stack

Use the functions in morphology method/laplacian_sobel_proj.py to either make the Laplacian or the Sobel projections of the original stacks.

Laplacian and Sobel filters

Run laplacian_filter.py to produce our Laplacian filter on stacks. folder_path should point to a directory containing only the stacks that need the filter applied. The path on line 39 should be change to the desired location of saving the resulting stacks.
To produce Sobel filters instead, use sobel_filter.py.

Morphology

To run our morphological models, use the corresponding file (2d_morph_1.py, 2d_morph_6_11_gen.py, layerwise_morph_1.py, layerwise_morph_6_11.py or layerwise_morph_all.py). No matter which of these you wish to run, the variable folder_path should point to a directory containing only the desired input images. For layerwise_morph_1.py and layerwise_morph_all.py that is Laplacian stacks produced by laplacian_filter.py for layerwise_morph_6_11.py that is Sobel stacks as produced by sobel_filter.py and for the rest that is a weighted Sobel projection as produced by method/laplacian_sobel_proj.py.
The paths for saving the resulting files should also be changed as desired.

Evaluate segmentations

To run our evaluation of the masks with the file Evaluation/evaluate_mask.py one has to change the paths on line 165 and 166 to the correct folders. If a 2D-masks is being evaluated, the line at 170 must be uncommmented, and in for 3D-masks.