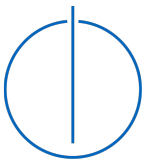
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Light Microscopy Image Segmentation using Gaussian Processes

Zayneb Miri



DEPARTMENT OF INFORMATICS

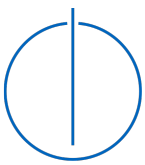
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Light Microscopy Image Segmentation using Gaussian Processes

Segmentierung von Lichtmikroskopiebildern mit Gauss-Prozessen

Author:	Zayneb Miri
Supervisor:	Prof. Dr. Christian Mendl
Advisor:	Dr. Felix Dietrich
Submission Date:	15.05.2022



I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.05.2022 Zayneb Miri

Acknowledgments

First, I would like to thank my advisor Dr.Felix Dietrich and my supervisor Prof. Christian Mendl for allowing me to write my bachelor's thesis at the Chair of Scientific Computing in Computer Science. A special thanks to Felix for guiding me through the journey by receiving invaluable feedback.

Secondly, Thank you to the best cosine in the world, Wiwi, for supporting me emotionally and mentally. A big thanks to Eli, Macha, and Bibi for your friendship and support.

Finally, a special thanks to my parents and siblings for the guidance, love, and everyday support.

I am grateful for all teachers, Professors that believed in me. I would not be here writing this thesis without you.

Abstract

In computer vision (CV), image segmentation is one of the most critical problems. A wide range of applications, including medical imaging and robotics, make it highly appealing. One of the applications where computer vision can help automate human labor is the segmentation of microscopy images. As deep learning models have gained considerable traction across a range of vision applications, there has been much research into building deep learning approaches to image segmentation. Convolutional Neural Networks have succeeded and are widely used, particularly for computer vision tasks.

Compared with state-of-the-art classifiers, the Gaussian process classifier offers several noticeable advantages. For example, its Bayesian nature allows any prior information to be used in the classification process. They also yield a posterior probability estimate and a variance estimate, which can be exploited as a confidence value for the predicted decision. By definition, a Gaussian process (GP) is a distribution over functions determined by its mean and kernel function. Therefore, it is essential for GP modeling to select an appropriate kernel function for a particular problem. Surprisingly, there is a direct connection between convolutional networks and Gaussian processes with specifically designed kernels. These kernels can be constructed in the limit of infinite kernels of the original convolutional networks.

The first aspect of this thesis will explain Gaussian processes, how they can be deduced from Convolutional Neural Networks, and how they can be used for classification. As segmentation is a form of labeling in which every pixel is assigned to a label, we will first train a Gaussian Process Classifier to perform pixel-level classification and then use the classifier to generate semantic segmentations. First, to illustrate the GP-CNN kernel's capability, we perform semantic segmentation of several images from the MNIST data set. Then, we focus on the light microscopy images. As a result, we found that contrary to the segmentation of MNIST images, the Gaussian process performed poorly on microscopy images.

Contents

Acknowledgments **iii**

Abstract **iv**

1 Introduction **1**

2 Related Work **3**
2.1 Arbuscular Mycorrhiza Fungi 3
2.2 3D Model of fungus-colonized root section 4
2.3 Gaussian Processes 6
2.3.1 Multivariate Gaussian distribution 6
2.3.2 Definition of Gaussian Processes 8
2.3.3 Gaussian Processes for Regression 8
2.3.4 Gaussian Process for Classification 10
2.4 Neural Networks as Gaussian Processes 10
2.4.1 Convolutional Neural Networks 10
2.4.2 Convolutional kernel 12
2.4.3 ReLU activation 13
2.5 Image Segmentation 13
2.5.1 Introduction to Image Segmentation 13
2.5.2 State-of-the-art of Image Segmentation 14

3 Light Microscopy Image Segmentation using Gaussian Processes **17**
3.1 Light Microscopy Data Set 17
3.1.1 Building the patches data set 17
3.1.2 Data Pre-processing 19
3.2 Our Approach 19
3.3 Evaluation metrics 20
3.4 Implementation 22
3.5 Performance Evaluation 23
3.5.1 MNIST Results 23
3.5.2 Microscopy Data Set Results 25

3.5.3 Visual results 30

4 Conclusion **41**

List of Figures **42**

List of Tables **44**

Bibliography **46**

1 Introduction

In biomedical applications, microscopy images help researchers analyze genetic perturbations, seek drug discovery, and phenotype cells [2]. Nevertheless, manual microscopy image analysis can be a complicated process that is susceptible to human error. On the other hand, machine Learning can segment, recognize specific objects of interest, and learn patterns in images to collect quantitative information. Consequently, image analysis could be optimized and expedited by applying machine learning techniques. For instance, Achim Hekler et al. [9] proved that automated image analysis employing Convolutional Neural Networks (CNNs) outperforms domain experts in classifying histopathological melanoma images. Image segmentation is currently one of the most challenging tasks of microscope image analysis since it is time-consuming to generate ground truth labels manually [13].

Furthermore, neural networks have the drawback that they require large training data set to generalize well [23]. In applications such as light microscopy image segmentation, where availability of training data is minimal, kernel-based methods are the recuse since they need fewer data to train. In this thesis, we propose a semantic segmentation approach using a Bayesian kernel method. The kernel method called Gaussian process derived from a CNNs in the limiting case of infinitely many channels. The GP-CNN kernels achieve state-of-the-art image classification results [6]. Therefore, our approach is based on pixel-wise classification, where we deploy a Gaussian process classifier to perform the task.

The thesis is organized as follows: In Chapter 2, we will introduce state of the art. It consists of five parts. In part 1, we will explain the biological background of Arbuscular Mycorrhiza Fungi (AMF). We will describe the development of the render pipeline that will be used to generate synthetic microscopy data in part 2. In part 3, we will explain Gaussian processes. In part 4, we will explain the equivalence between Gaussian processes and Convolutional Neural Networks (CNNs). In the last part of the chapter, we will define what image segmentation is and review some image segmentation techniques.

In the main part of the thesis, chapter 3, we will start by describing the steps of building the light microscopy data set. Next, we will explain our Gaussian processes

classification approach used to predict the image segmentation maps. Afterward, we will present the used evaluation metrics and give implementation details, where we will mention which libraries are used and outline the technical environment of the machines used for the training. Subsequently, we will provide an overview of the performance of our approach on the MNIST and light microscopy data sets.

Finally, in chapter 4, we will overview the thesis and highlight our key findings. Then, we will give suggestions for future work.

2 Related Work

This thesis aims to perform semantic segmentation on microscopy images using Gaussian processes. This chapter contains an explanation of the biological background and how the synthetic images of the data set are created. Then we introduce Gaussian processes and how they can be used for supervised machine learning tasks such as regression and classification. Lastly, we provide an overview of the methods employed to solve the image segmentation task.

2.1 Arbuscular Mycorrhiza Fungi

Arbuscular mycorrhiza fungi (AMF) are soil microorganisms that symbiotically interact with 90% of agricultural plants and 80–90% of vascular plant species [24].

Fungal hyphae are long filamentous branches that form arbuscules enclosed by the plant cell's plasma membrane, develop inside the root, and pierce its cell walls to create arbuscules. These arbuscules facilitate nutrient transfer by creating a more excellent contact surface between the fungus and the plant. The fungus helps the nutrient transfer with the uptake of nitrogen, water, phosphorus, and amino acids. In return, the plants furnish carbon [30].

The AMF development process can be decomposed into a few steps: first, the symbiotic partners exchange molecules for mutual recognition. The fungus is then guided into deeper cell layers by an intracellular accommodation structure formed by the contacted plant cells. Ultimately, the fungal hyphae move lengthwise through the root cortex's gaps between the cells. These hyphae generate other branches, which leads to the creation of arbuscles within plant cells. The distinction of vesicles that store oil-rich products are part of the post-arbuscular evolution [30]. A visualization of AMF is shown in Figure 2.1. Arbuscules, hyphae, and vesicles are the objects that have to be segmented with our approach in section 3.2.

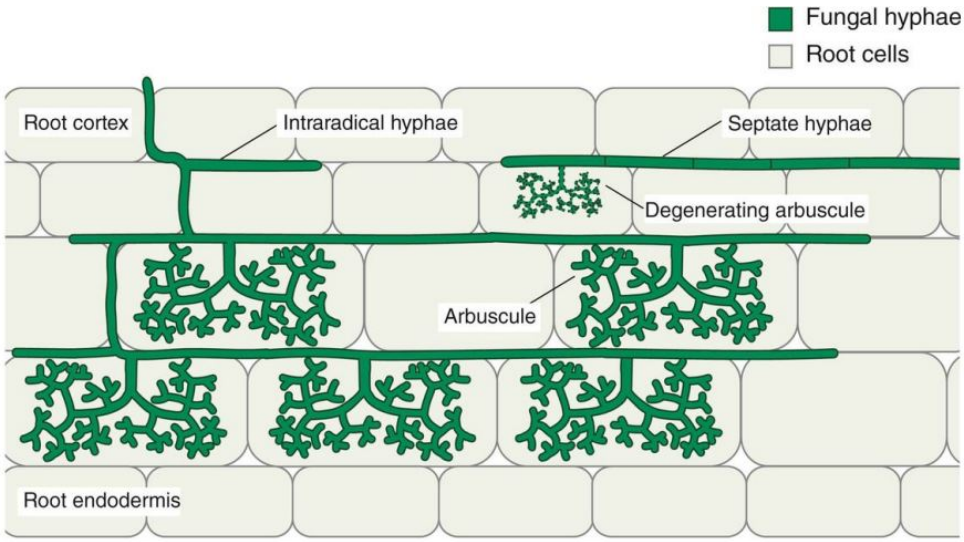


Figure 2.1: AMF structure, taken from [30].

2.2 3D Model of fungus-colonized root section

To prepare the AMF images, the *Lotus japonicus ecotype Gifu* plant seeds were first scarified and surface sterilized. Then, these imbibed seeds were grown for 10 to 14 days before being cultivated in quartz sand as a substrate. Next, the roots were inoculated with 500 spores per plant for colonization with the fungus *Rhizophagus irregularis* and collected five weeks later. Afterward, the fungal structures were stained to make them visible under the microscope. Finally, a Leica DM6 B wide-field microscope was used to image colonized root sections. The obtained images can then be pixel-wise annotated by a plant biologist [30], as shown in figure 2.2.

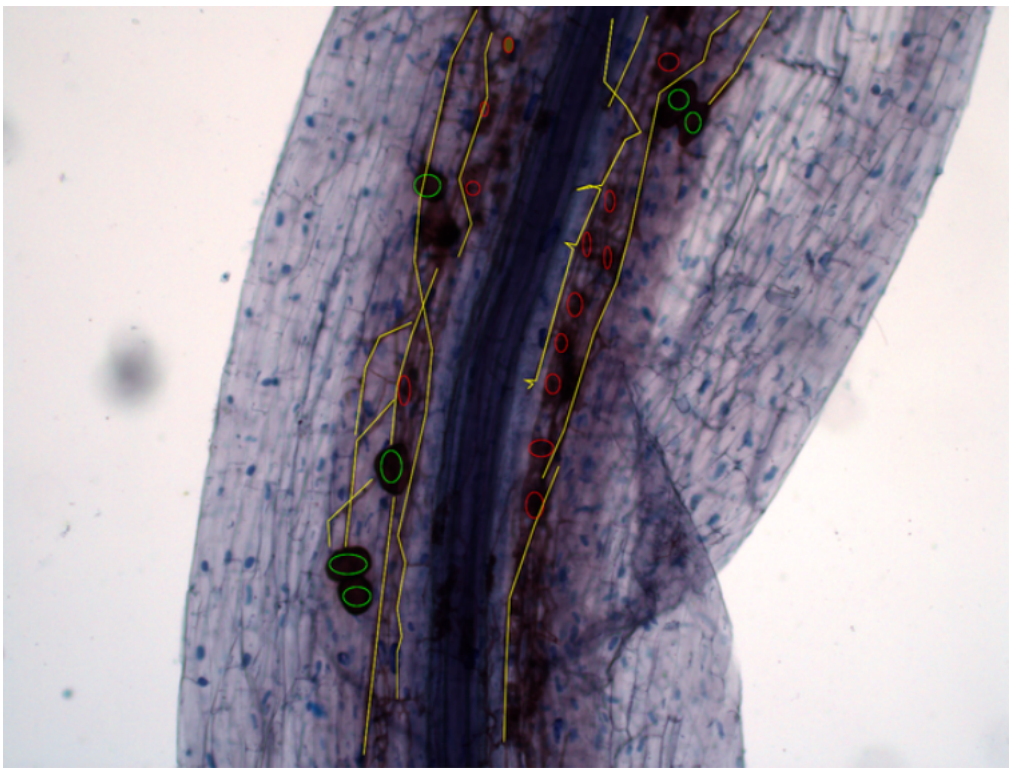


Figure 2.2: Annotated colonized root section. Hyphae are annotated with yellow lines, arbusclues with red circles, and vesicles with green circles, taken from [30].

Since it is challenging to obtain ground truth pixel-wise annotations, we will use generated synthetic images for our experiments. Jan Watter [30] used a 3D model of the fungus-colonized root section developed using the software Blender. We will describe the simulation pipeline shortly. It was broken down into various stages: object modeling, rigging, shading, compositing, and rendering. Initially, the object, which is, in this case, the plant root anatomy, was modeled. The model consists of a surface representation (a mesh) until now. Hence, the ringing technique was applied to add bones (a set of interconnected parts) that can deform the 3D mesh. The generation of new configurations was done using the blender Python API. A python script was created, which produced different configuration files and ran the blender rendering on them. Employing a random number of configurations, the model's mesh will be slightly rotated, scaled, and translated for each rendering process. Then materials, the background of the scene, and lights (shading nodes) were used to build a network of connected nodes, and the renderer then used their resulting output to create the

final rendered image and the corresponding segmentation map. Figure 2.3 shows an example of render and its corresponding segmentation map.

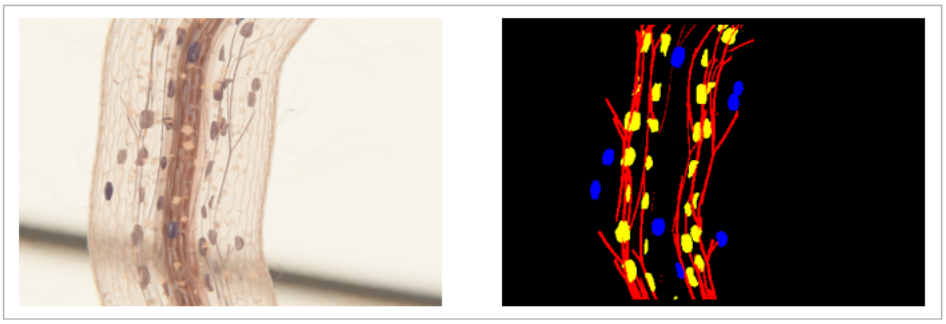


Figure 2.3: Render image on the left, segmentation map on the right. Vesicles (blue), arbuscules (yellow) and hyphae (red).

2.3 Gaussian Processes

In this section, we first make the mathematical intuition behind Gaussian processes more approachable 2.3.1. Secondly, we explain what a Gaussian process is 2.3.2. Finally, we clarify how we use Gaussian processes for regression 2.3.3 and classification 2.3.4.

2.3.1 Multivariate Gaussian distribution

Definition 2.3.1 (Gaussian Process) *A Gaussian process is a generalization of the Gaussian probability distribution. Whereas a probability distribution describes random variables which are scalars or vectors (for multivariate distributions) [21].*

From the definition above 2.3.1, we note that before exploring Gaussian processes, there are underlying mathematical concepts we need first to understand.

Each random variable is distributed normally in a multivariate distribution, and their joint distribution is Gaussian. A multivariate Gaussian distribution is defined by a mean vector $\boldsymbol{\mu}$ and a covariance matrix $\boldsymbol{\Sigma}$. For $\mathbf{X} = [\mathbf{X_1}, \ldots, \mathbf{X}_n]^T$ set of random variables, we say X follows a normal distribution:

$$X \sim \mathcal{N}(\mu, \Sigma)\,. \tag{2.1}$$

The diagonal of Σ refers to the variance σ_i^2 of the i-th random variable and the off-diagonal elements $\sigma_{i,j}^2$ corresponds to the variance between the i-th and j-th random variable [5].

To explain two essential concepts, the marginal and conditional distribution, we consider a simple example, a random vector $Z = [X, Y]^T$ where X and Y are subsets of Gaussian jointly random variables, with parameters:

$$P_{X,Y} \sim \mathcal{N}(\mu, \Sigma) = \mathcal{N}\left(\begin{bmatrix}\mu_X\\ \mu_Y\end{bmatrix}, \begin{bmatrix}\Sigma_{XX} & \Sigma_{XY}\\ \Sigma_{YX} & \Sigma_{YY}\end{bmatrix}\right). \tag{2.2}$$

Marginal Distribution

A multivariate probability distribution can be marginalized to extract partial information. Every marginal distribution of a Gaussian distribution is itself gaussian given by:

$$\begin{aligned} X &\sim \mathcal{N}(\mu_X, \Sigma_{XX}),\\ Y &\sim \mathcal{N}(\mu_Y, \Sigma_{YY}). \end{aligned} \tag{2.3}$$

More specifically, given the above normal probability distribution, the marginal distribution of X and Y is defined as follows:

$$\begin{aligned} p_X(x) &= \int_y p_{X,Y}(x,y)\ \mathrm{d}y = \int_y p_{X|Y}(x \mid y) p_y(y)\ \mathrm{d}y,\\ p_Y(y) &= \int_x p_{Y,X}(y,x)\ \mathrm{d}x = \int_x p_{Y|X}(y \mid x) p_x(x)\ \mathrm{d}x. \end{aligned} \tag{2.4}$$

Conditional Distribution

We use conditional distribution to determine the probability of a variable depending on another variable. Every conditional distribution of a Gaussian distribution is again Gaussian, and it can be defined as follows:

$$\begin{aligned} P_{X|Y} &\sim \mathcal{N}\left(\mu_X + \Sigma_{XY}\,\Sigma_{YY}^{-1}(Y - \mu_Y), \Sigma_{XX} - \Sigma_{XY}\,\Sigma_{YY}^{-1}\,\Sigma_{YX}\right),\\ P_{Y|X} &\sim \mathcal{N}\left(\mu_Y + \Sigma_{XX}\,\Sigma_{YX}^{-1}(X - \mu_X), \Sigma_{YX} - \Sigma_{YX}\,\Sigma_{XX}^{-1}\,\Sigma_{YY}\right). \end{aligned} \tag{2.5}$$

The four equations in 2.4 and 2.5 are driven from [16].

2.3.2 Definition of Gaussian Processes

A Gaussian Process is a set of random variables, of which any finite subset is jointly Gaussian distributed [21]. Like Gaussian distributions, a Gaussian process is fully specified by mean and covariance function.

Let $f(x)$ be a GP with $m(x)$ the mean function and $k(x, x^{'})$ covariance function, also called kernel function. Then we express a GP as follows:

$$\boldsymbol{f(x) \sim \mathcal{GP}\left(m(x), k(x, x^{'})\right)} \tag{2.6}$$

where,

$$\begin{aligned} m(x) &= \mathbb{E}[(f(x))] \text{ and} \\ k(x, x^{'}) &= \mathbb{E}[(f(x) - m(x))(f(x^{'}) - m(x^{'}))]. \end{aligned} \tag{2.7}$$

Gaussian processes can represent distributions over functions. So, in this case, each random variable in a Gaussian process could stand for a function value at a different input value.

Consider a linear regression model $f(x) = \phi(x)^T w$ with prior $w \sim \mathcal{GP}(0, \Sigma_p)$. We can calculate its mean and covariance with equation 2.7 as follows:

$$\begin{aligned} \mathbb{E}[f(x)] &= \phi(x)^T \mathbb{E}[w] \text{ and} \\ \mathbb{E}[f(x)f(x^{'})] &= \phi(x)^T \mathbb{E}[ww^T]\phi(x^{'}) = \phi(x)^T \Sigma_p \phi(x^{'}). \end{aligned} \tag{2.8}$$

To sum up, the function output for two inputs, $f(x)$ and $f(x^{'})$, are jointly Gaussian distributed with mean 0 and covariance $\phi(x)^T \Sigma_p \phi(x^{'})$. Function values for all inputs are related in this way.

As a well-known example of the kernel function, we represent the square exponential (SE), which is given by

$$cov(f(x), f(x^{'})) = k(x, x^{'}) = exp\left(-\frac{1}{2}| \ x - x^{'} \ |^2\right). \tag{2.9}$$

SE covariance is smaller when input values are farther apart and more significant when input values are closer together.

2.3.3 Gaussian Processes for Regression

The Gaussian process regression is a supervised learning method in which we seek a function f that relates the input to the output. For example, let us assume that f is

distributed with a zero-mean Gaussian process prior and SE covariance, a training data set X for which we know the output values, and a testing data set X^* for which the output values are unknown. The aim is to predict the test outputs f^* for the testing dataset X^* by combining the prior and the knowledge provided by the training dataset about the function. We denote the joint distribution of f and f^* as:

$$\begin{bmatrix} f \\ f^* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X,X) & K(X,X^*) \\ K(X^*,X) & K(X^*,X^*) \end{bmatrix}\right). \tag{2.10}$$

This equation represents a prior since it does not reflect information learned from the training data, where:

- $K(X,X)$: covariance matrix over the entries of X.
- $K(X^*,X)$ and $K(X,X^*)$: covariance matrix between entries of X and X^*.
- $K(X^*,X^*)$: covariance matrix over the entries of X^*.

Marginalizing out the points where the targets are given, we compute the conditional posterior distribution as described in 2.5, we get:

$$f^* \mid X^*, X, f \sim \mathcal{N}(K(X^*,X)K(X,X)^{-1}f, \\ K(X^*,X^*) - K(X^*,X)K(X,X)^{-1}K(X,X^*)). \tag{2.11}$$

The posterior distribution in equation 2.11 estimates function values for the unseen data points. The mean $K(X^*,X)K(X,X)^{-1}f$ corresponds, in fact, to the most likely value. Besides, the standard deviation values found in the covariance matrix can quantify how confident we are in the predictions. A lower standard deviation indicates a narrower overall distribution, and we may be more confident about our predictions.

In more realistic situations, the data set includes noisy observations. Let $y = f(x) + \varepsilon$ be the noisy observations with variance σ_n^2, where ε is the added Gaussian noise. Consequently, we can rewrite the join distribution of functions values for training and testing points as follows:

$$\begin{bmatrix} y \\ f^* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X,X) + \sigma_n^2 I & K(X,X^*) \\ K(X^*,X) & K(X^*,X^*) \end{bmatrix}\right). \tag{2.12}$$

Likewise, we can derive a new distribution conditioned corresponding to equation 2.5 on the information provided in y to get the predictive equations for Gaussian process regression as:

$$f^* \mid X, y, X^* \sim \mathcal{N}(K(X^*,X)[K(X,X) + \sigma_n^2 I]^{-1}y, \\ K(X^*,X^*) - K(X^*,X)[K(X,X) + \sigma_n^2 I]^{-1}K(X,X^*)). \tag{2.13}$$

2.3.4 Gaussian Process for Classification

A regression model predicts a single output given an input, while a Multi-output regression predicts numerous outcomes to a given input. The target is binary in a classification model, whereas it is real-valued in a regression model [26]. Section 2.3.3 explained GP regression, which has a real-valued output. Thereby, we need to transform the classification task to a multi-output regression task as performed by Adrià Garriga-Alonso [6]. For a given training data set X and the corresponding labels vector Y, the predicted labels for the unseen data set X^*, corresponds to a row-wise maximum of $\mathbf{K}(\mathbf{X}^*, \mathbf{X})\mathbf{K}(\mathbf{X}, \mathbf{X})^{-1}\mathbf{Y}$ [6].

Another option would be to use multi-class classification with Gaussian processes directly. Nevertheless, the non-Gaussian likelihood has to be approximated using techniques such as Laplace approximation, which is time-consuming [21].

2.4 Neural Networks as Gaussian Processes

There are numerous and reasonable kernel functions. However, we will focus on the GP kernel induced by a Convolutional Neural Network (CNN). Adrià Garriga-Alonso showed the equivalence between CNN and GPs in the limit of infinitely many convolutional kernels, and the proof can be studied in [6]. In this section, we explain CNNs (2.4.1). Then we represent the mean and covariance function of the convolutional kernel (2.4.2). Ultimately we present the ReLU activation (2.4.3).

2.4.1 Convolutional Neural Networks

CNN differs from other classical neural networks in that the input is a multichannel image rather than a vector format [7]. Given an input image X for the CNN with height $H^{(0)}$, width $D^{(0)}$, and channels $C^{(0)}$ ($C^{(0)} = 1$ for gray-scale image and 3 for RGB images). X is represented as a matrix of the size $C^{(0)} \times (H^{(0)}D^{(0)})$. Each row $x_1, x_2, ..., x_{C^{(0)}}$ of the matrix corresponding to a channel of the image is flattened to a one-dimensional vector. We will consider a convolutional neural network with L hidden layers. The first layer transforms the input image linearly. For $i \in 1, ..., C^{(1)}$:

$$a_i^{(1)}(X) := b_i^{(1)}1 + \sum_{j=1}^{C^{(0)}} W_{i,j}^{(1)} x_j. \tag{2.14}$$

Figure 2.4 shows a 2D convolutional $U_{i,j}^{(0)} * x_j$ that can be expressed as matrix multiplication. $U_{i,j}^{(0)}$ is transformed to $W_{i,j}^{(0)}$ by taking the 2×2 kernel filter $U_{i,j}^{(0)}$ and sliding it horizontally and vertically over the input image's jth channel, and x_j is flattened into a vector. By applying the filter to the μ'th convolutional patch of the input image's j'th channel x_j, we obtain j'th row of $W_{i,j}^{(0)}$. The empty positions of $W_{i,j}^{(0)}$ are zeros where the filter is not applied. Of course, in the case of multiple channels, the convolutional operation has to be applied to each channel, and then the result of matrix multiplication has to be summed up $\sum_{j=1}^{C^{(0)}} W_{i,j}^{(1)} x_j$. In the end, a bias term $b_i^{(l)}$ is added [6].

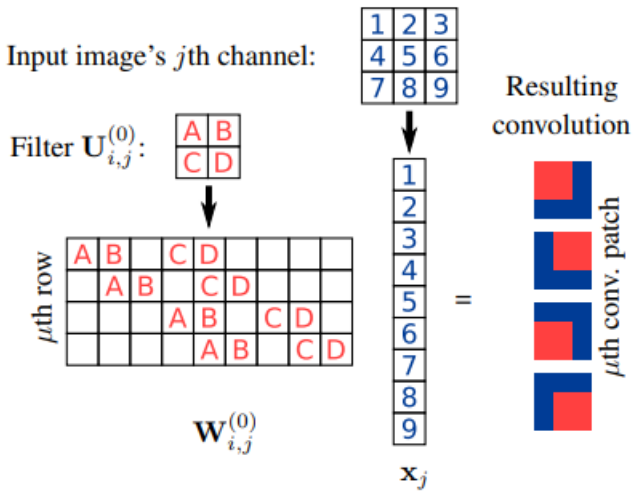


Figure 2.4: 2D Convolutional, where the kernel $U_{i,j}^{(0)}$ is applied to an input image x_j, taken from [6].

The layers that follow are applied to the created feature maps from the previous layer. Thus, we define feature maps of the layers as:

$$a_i^{(l+1)}(X) := b_i^{(l+1)} 1 + \sum_{j=1}^{C^{(l)}} W_{i,j}^{(l+1)} \phi(a_j^l(X)). \tag{2.15}$$

Each row of $a_i^{(l+1)}$ is the flattened j'th channel of the image after the convolutional filter was applied to $\phi(A^l(X))$ with $A^l(X)$ of a shape $C^{(l)} \times (H^{(l)} D^{(l)})$ where, $i \in 1, ..., C^{(l+1)}$, and $j \in 1, ..., C^{(l)}$. The output of CNN is the last feature map $A^{l+1}(X)$. In the case of classification or regression, we have $H^{l+1} = D^{l+1} = 1$. Consequently, the pseudo-weights have one row, and the resulted activations $W_{i,j}^{(L+1)}$ are a one-element vector [6].

2.4.2 Convolutional kernel

for the above defined convolutional neural network lets the weight and biases be normally distributed and denoted as follows:

$$\begin{aligned} \mathrm{U}_{i,j,x,y}^{(l)} &\sim \mathcal{N}\left(0, \sigma_{\mathrm{w}}^2 / C^{(l)}\right) \text{ and} \\ \mathrm{b}_i^{(l)} &\sim \mathcal{N}\left(0, \sigma_{\mathrm{b}}^2\right). \end{aligned} \tag{2.16}$$

The CNN from now on will be expressed as:

$$\mathrm{A}_{i,\mu}^{(l+1)}(\mathrm{X}) = \mathrm{b}_i^{(l+1)} + \sum_{j=1}^{C^{(l)}} \sum_{\nu=1}^{H^l D^l} \mathrm{W}_{i,j,u,\nu}^{(l+1)} \phi(\mathrm{A}_{j,\nu}^{l}(\mathrm{X})), \tag{2.17}$$

where l and $l+1$ are the input, and output layers, i and $j \in 1, ..., C^{(l+1)}$ denote the input and output channels, and u and $v \in 1, ..., C^{(l+1)}$ index the location within the feature maps.

Mean function of the convolutional kernel

The mean function is equal to zero since the weights and the biases have zero mean, and last but not least, the weights are independent of the results of the previous layer

$$\mathbb{E}[\mathrm{A}_{i,\mu}^{l+1}(\mathrm{X})] = \mathbb{E}[\mathrm{b}_i^{l+1}] + \sum_{j=1}^{C^{(l)}} \sum_{\nu=1}^{H^l D^l} \mathbb{E}[\mathrm{W}_{i,j,u,\nu}^{(l+1)} \phi(\mathrm{A}_{j,\nu}^{l}(\mathrm{X}))] = 0. \tag{2.18}$$

Covariance function of the convolutional kernel

The covariance function between two different inputs, X and X$'$ can be computed as follows:

$$\begin{aligned} \mathbb{C}\left[\mathrm{A}_{i,\mu}^{(l+1)}(\mathrm{X}), \mathrm{A}_{i,\mu}^{(l+1)}(\mathrm{X}')\right] = \sigma_{\mathrm{b}}^2 + \\ \sum_{j=1}^{C^{(l)}} \sum_{j'=1}^{C^{(l)}} \sum_{v=1}^{H^{(l)}D^{(l)}} \sum_{\nu'=1}^{H^{(l)}D^{(l)}} \mathbb{E}\left[\mathrm{W}_{i,j,\mu,\nu}^{(l+1)} \mathrm{W}_{i,j',\mu,\nu'}^{(l+1)}\right] \mathbb{E}\left[\phi(\mathrm{A}_{j,\nu}^{(l)}(\mathrm{X})) \phi(\mathrm{A}_{j',\nu'}^{(l)}(\mathrm{X}'))\right]. \end{aligned} \tag{2.19}$$

Since the weights $\mathrm{W}_{i,j}^{l+1}$ and $\mathrm{W}_{i,j'}^{l+1}$ are independent for $j \neq j'$. Additionally, each row μ of $\mathrm{W}_{i,j}^{l+1}$ has only zeros or independent values, as illustrated in figure 1 for $\nu \neq \nu'$. Hence, we can simplify 2.19 to:

$$\mathbb{C}\left[\mathrm{A}_{i,\mu}^{(l+1)}(\mathrm{X}), \mathrm{A}_{i,\mu}^{(l+1)}(\mathrm{X}')\right] = \sigma_{\mathrm{b}}^2 + \sum_{j=1}^{C^{(l)}} \sum_{v=1}^{H^{(l)}D^{(l)}} \mathbb{E}\left[\mathrm{W}_{i,j,\mu,\nu}^{(l+1)} \mathrm{W}_{i,j,\mu,\nu}^{(l+1)}\right] \mathbb{E}\left[\phi(\mathrm{A}_{j,\nu}^{(l)}(\mathrm{X})) \phi(\mathrm{A}_{j,\nu}^{(l)}(\mathrm{X}'))\right]. \tag{2.20}$$

From here, the sum over ν can be limited over the non-zeros values, for the reason that for the μ'th patch, the weights $W_{i,j}^{(l+1)}$ are zeros at the locations ν, which do not belong to the μ'th patch. The variance of the first layer can be expressed as follows:

$$K_{\mu}^{(1)}(X, X') = \mathbb{C}\left[A_{i,\mu}^{(1)}(X), A_{i,\mu}^{(1)}(X')\right] = \sigma_b^2 + \frac{\sigma_w^2}{C^{(0)}} \sum_{i=1}^{C^{(0)}} \sum_{\nu \in \mu th \ patch} X_{i,\nu} X'_{i,\nu}. \tag{2.21}$$

For any other layer, the variance can be described as follows:

$$K_{\mu}^{(1+1)}(X, X') = \mathbb{C}\left[A_{i,\mu}^{(1+1)}(X), A_{i,\mu}^{(1+1)}(X')\right] = \sigma_b^2 + \sigma_w^2 \sum_{\nu \in \mu th \ patch} V_{\nu}^{l}(X, X'), \tag{2.22}$$

where

$$V_{\nu}^{l}(X, X') = \mathbb{E}\left[\phi(A_{j,\nu}^{(l)}(X))\phi(A_{j,\nu}^{(l)}(X'))\right] \tag{2.23}$$

is the covariance of the activations.

2.4.3 ReLU activation

We need to compute equation 2.23 in a closed form with an activation function. For instance, for ReLU activation, we obtain the closed form of equation 2.23 as:

$$V_{\nu}^{l}(X, X') = \frac{\sqrt{K_{\nu}^{(l)}(X, X) K_{\nu}^{(l)}(X', X')}}{\pi} \left(\sin \theta_v{}^{(l)} + (\pi - \theta_v^{(l)}) \cos \theta_v{}^{(l)}\right). \tag{2.24}$$

2.5 Image Segmentation

In this section, we explain image segmentation (2.5.1), followed by a brief description of existing image segmentation techniques (2.5.2).

2.5.1 Introduction to Image Segmentation

We can group image segmentation tasks into semantic, instance, and panoptic segmentation categories. Semantic segmentation assigns each image pixel to a label such as a car, person, tree... It treats different objects of the same class as a single entity. Contrary to instance segmentation treats the objects with the same class label as individual other instances. Panoptic segmentation assigns two labels, a semantic label and an instance

id, to each pixel. In this manner, it is simply a combination of semantic and instance segmentation. Figure 2.5 shows an example of the three different above explained segmentation techniques.

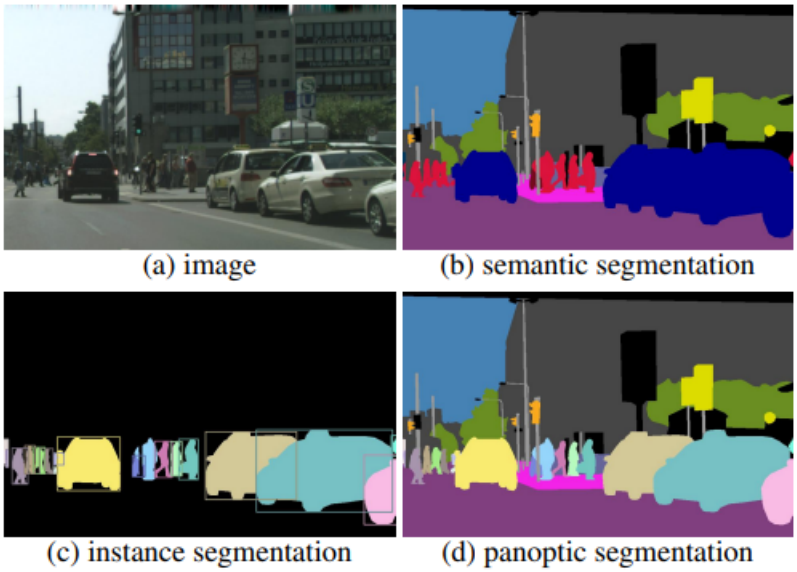


Figure 2.5: Example of a semantic, instance and panoptic segmentation. Taken from [12].

2.5.2 State-of-the-art of Image Segmentation

Before the coming of deep learning, we used classical methods such as modified SVM [31] and k-clustering [11] to solve the image segmentation task. Regardless, deep learning models achieved significant performance gains compared to existing methods. So as next, we will review these deep learning methods.

Fully Convolutional Network (FCN)

FCN is one of the primary deep learning approaches proposed by Jonathan Long [15] (Figure 2.6) for solving semantic segmentation challenges. To deal with the problem of inputs of varying sizes, the author adapted CNN architectures such as Alex-Net and VGG16. By replacing all fully connected layers with fully convolutional layers. As a result, the model outputs a spatial segmentation map of the input image instead of a classification score. After going through convolutional and max-pooling layers, the output is smaller than the input image size. Thus upsampling is applied to make

the output have the same size as the input. However, it makes the output label map rough. Since semantic information is obtained from deeper layers and spatial location information from fine layers, fusing the output after up-sampling and the output of the previous layer enhances the segmentation results. The model was tested on PASCAL VOC, SIFT flow, and NYUDv2 data sets and showed a performance improvement compared to convnets. Unfortunately, this model is not tested on light microscopy images to the best of our knowledge.

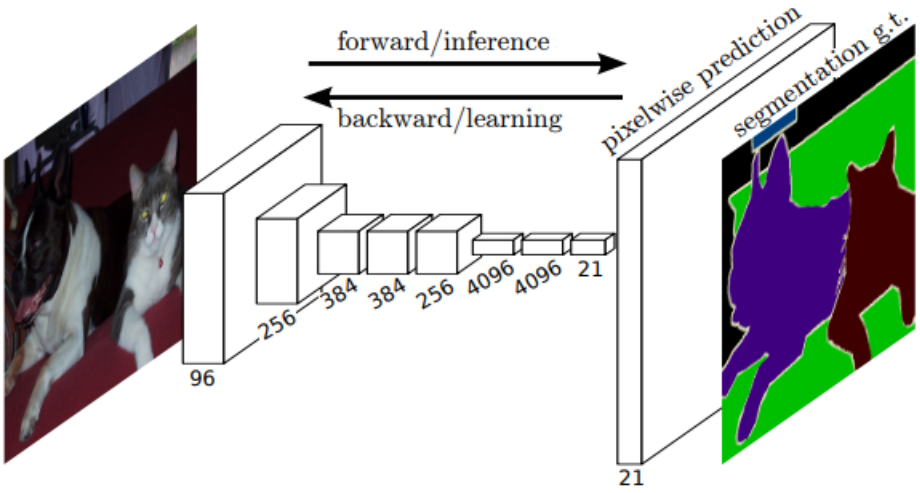


Figure 2.6: A Fully Connected Network that learns pixel-wise predictions, taken from [15].

Encoder-Decoder-Based Models

A well-known model is the U-Net, which is proposed by Olaf Ronneberger [22] built upon FCN for segmenting biological microscopy images. It consists of two parts contracting path (encoder) that encodes the input image to a feature representation and an expansive path (decoder) that project the features learned by the encoder onto the pixel space to get the final pixel-wise classification. Compared to FCN, U-net has multiple up-sampling layers instead of one layer. Also, fusing the output of earlier layers is done as concatenation rather than element-wise addition. Another popular model in this category is called Seg-Net. It is proposed by Vijay Badrinarayanan [1] (Figure 2.7). Thanh Tran et al. [28] used Seg-Net to segment white blood cells and red blood cells from the background of peripheral blood smear images, showing 89% global accuracy.

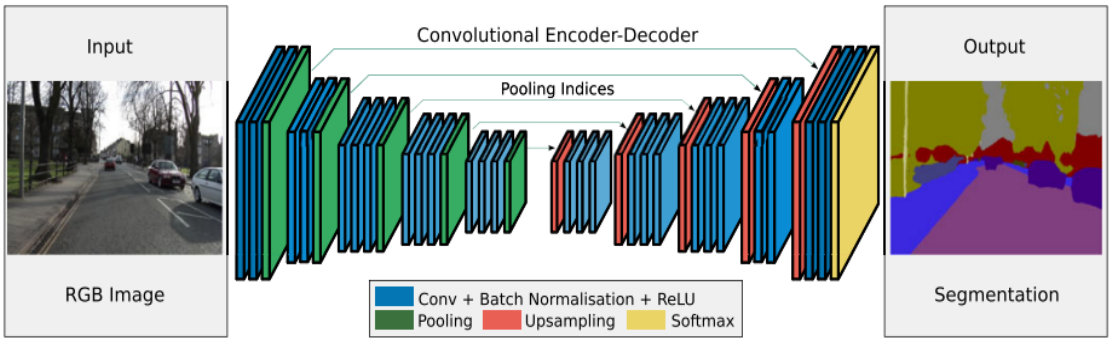


Figure 2.7: SegNet; The encoder applies convolutions and max pooling. The decoder up-samples the encoder output using the max pooling indices. Finally the softmax layer performs pixel-wise prediction. Taken from [1].

Gaussian Processes Based Models

As mentioned before, Gaussian Processes provide an uncertainty estimation for the prediction. For this reason, some researchers use this property of GPs to achieve state-of-the-art results on image segmentation tasks.

For example, Sebastian G Popescu et al. [20] developed a GP-based convolutional architecture for segmentation of tumors in the brain using an MRI data set. Comparing this model with a U-Net, they demonstrated it achieved comparable results in image segmentation tasks.

Moreover, for few-shot segmentation (FSS), Joakim Johnander et al. [10] rather than using Gaussian process as the final output, suggests using Gaussian process regression (GPR) incorporated in an encoder-decoder architecture. First, training and testing images and the corresponding masks are fed through an encoder. Their corresponding features are then employed in Gaussian process regression to infer the distribution of the testing mask given the training set and testing images. The mean and variance of the GP predictive distribution are used as input for a CNN-based decoder. Thus, uncertainty is considered to create the final predictions segmentation. The model is tested on the PASCAL-5^i and COCO-20^i data sets, demonstrating that it outperforms other baselines.

3 Light Microscopy Image Segmentation using Gaussian Processes

In this chapter, we describe how we created the microscopy data set. Then we explain the Gaussian process-based approach used for semantic segmentation. Additionally, we introduce a summary of used evaluation metrics and implementation details. At last, we show performance evaluation on MNIST and microscopy images data set.

3.1 Light Microscopy Data Set

In this section, we show how we built the data set consisting of small patches. Then we illustrate the data pre-possessing steps we applied to the created data set to feed them as an input to the Gaussian process classifier.

3.1.1 Building the patches data set

The Image resolution is $[736 \times 973]$ for computational reasons to avoid memory overload. To further increase the number of training samples, we will extract multiple overlapping patches of a single image, as illustrated in figure 3.2. Each patch has a size of $[256 \times 256]$ because choosing a more petite size patch will allow the network to see only a little context. We will extract the same patches from the corresponding segmentation map image. Extracting a large number of patches is not enough. As well as the diversity of patches, it is necessary to ensure that we present all cases in a balanced proportion. Wherefore, we cut the image into patches with a step size of 128. We conducted some experiments on patches with a step size of 64.

First, we generate 45 synthetic images as outlined in section 2.2. Secondly, we applied zero padding on them to ensure that the height and width are indivisible with 256 and thus that the whole image is cropped. Afterward, we split them manually into testing, validation, and testing set, where we make sure that the images look different among the three sets. Subsequently, we create patches for each subset. Finally, we

cleaned them. By cleaning, we mean to remove the patches where only the background is included. Figure 3.1 demonstrate the steps we go through to create the final data set.

The algorithm for creating the patches using the framework [18] works as follows: it starts with the first patch on the top left corner of the image. It is then transferred to 128 or 64 pixels to the right, and it is continuously done until it reaches the right side of the image. This procedure is also repeated vertically until the patches reach the bottom of the image.

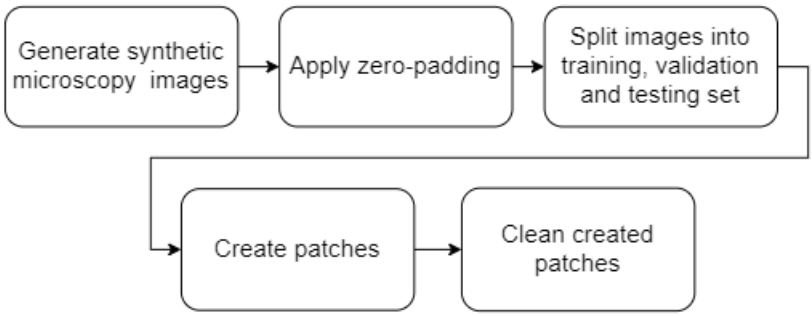


Figure 3.1: Illustration of the steps for creating the patches data set.

Figure 3.2: Split the image into overlapping patches. The blue, red, and green blocks show an example of three overlapping patches on the left—some patch samples on the right.

3.1.2 Data Pre-processing

After extracting the patches and building the training, validation, and testing subsets, we need to perform a few pre-processing steps to the segmentation maps and the images for the Gaussian Process Classifier (GPC) model. All the step details are depicted in figure 3.3. Finally, we fed the final output of the pre-processing steps to the model.

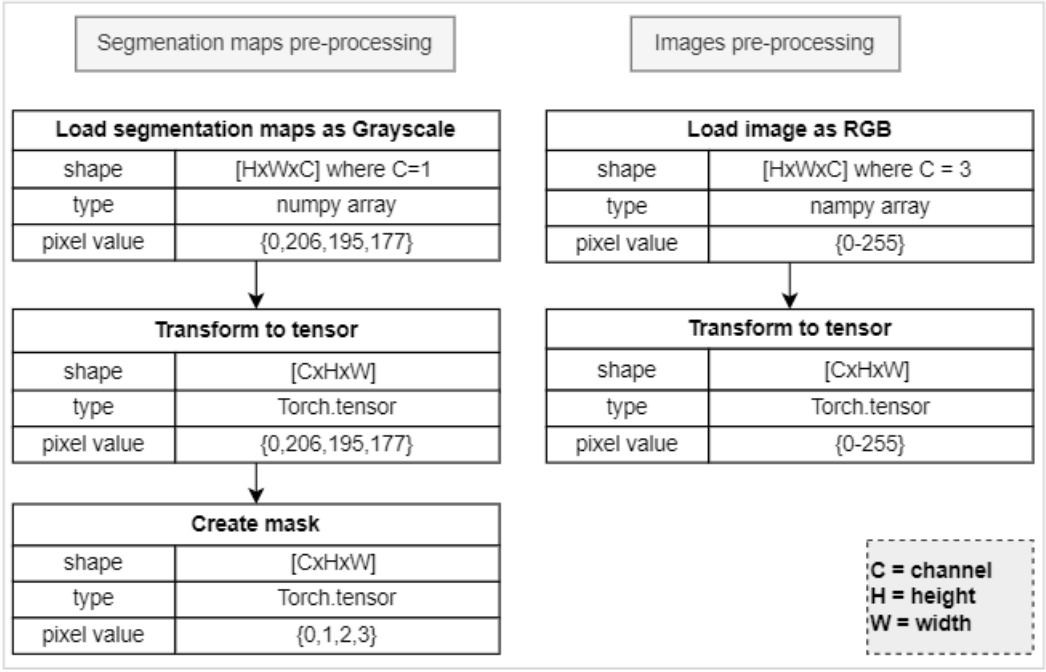


Figure 3.3: Flow diagram of pre-processing steps: the right side shows the transformations for the rendered image, the left side shows the transformations for the segmentation maps.

3.2 Our Approach

Given a training set $\mathcal{D}$ that contains N samples and their targets representing the corresponding segmentation maps, $\mathcal{S}$. The objective is to predict the segmentation masks of an unseen set of images $\mathcal{D}^*$ that contains M samples. In our approach, as illustrated in figure 3.4, we first pre-processed the data as described in subsection 3.1.2 before feeding them to the Gaussian Process Classifier (GPC). The GPC needs

three inputs as a tensor. We denoted $\mathcal{D}$, $\mathcal{D}^*$, and $\mathcal{S}$ after the pre-processing step by $X \in [N, C, H, W]$, $X^* \in [M, C, H, W]$, and $Y \in [N, C, H, W]$, respectively. We get the predicted segmentation masks as a tensor denoted by $Y^* \in [M, C, H, W]$ for the unseen images as an output.

We build a GPC, where the classification works as described in subsection 2.3.4. Informally explained, we will have n inferred input-output functions $f_{p_1}, \ldots, f_{p_n}$, where $p_1, \ldots, p_n$ refers to the pixels in an image. Each function is later used to map the $i'th$ pixel of an unseen image to a label, where $i \in 1, \ldots, n$.

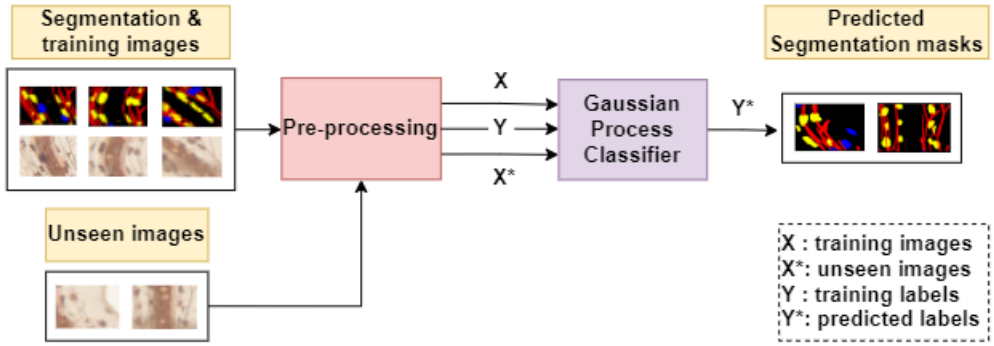


Figure 3.4: Overview of our approach: the training images, corresponding segmentation maps, and the unseen images set are first pre-processed and then fed into the Gaussian Process Classifier. Finally, we get the segmentation masks Y^* as output.

3.3 Evaluation metrics

When evaluating the performance of semantic segmentation models, we compare the ground truth with the predicted segmentation masks. Semantic segmentation is performed through pixel-wise classification. Hence, they are based on pixel-wise similarities. In this section, we will explain the following evaluation metrics:

- Recall
- Precision
- F1-score
- Pixel Accuracy
- Mean Pixel Accuracy
- Intersection over union
- Mean Intersection over union

Recall and **precision** metrics are computed for each class as follows:

$$Recall = \frac{TP}{TP + FN} \tag{3.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

Where TP are the true positives, FP the false positives and FN the false negatives. The recall is the true positive rate, and it is frequently used in image-based research [25] while precision represents the positive prediction value.

We define the following notations for the further presented metrics: X is the ground truth mask for an image, and Y is the predicted segmentation mask. n_{ij} refers to the number of pixels that corresponds to class i, classified as class j and, n_{ii} refers to the number of pixels that belong to class i and are predicted to belong to class i. At last, c refers to the total number of classes.

F1-score considers both the recall and precision. In particular, it is a harmonic mean of both metrics

$$F1\text{-}score = \frac{2TP}{2TP + FP + FN} \tag{3.3}$$

Pixel Accuracy is calculated by dividing the pixels that are correctly predicted by the total number of pixels:

$$PA = \frac{\sum_i n_{ii}}{\sum_i \sum_{j=1}^{c} n_{ij}} \tag{3.4}$$

Mean Pixel Accuracy is calculated per-class basis and then averaged over the classes:

$$MPA = \frac{1}{c} \sum_i \frac{n_{ii}}{\sum_{j=1}^{c} n_{ij}} \tag{3.5}$$

Intersection over Union is computed by dividing the number of common pixels between the ground truth and the predicted mask by the total number of pixels:

$$IoU\,(X, Y) = \frac{\mid X \cap Y \mid}{\mid X \mid + \mid Y \mid - \mid X \cap Y \mid} \tag{3.6}$$

Where $IoU \in [0, 1]$. Thus if IoU is equal to zero, when there is no similar pixel labeling between ground truth and predicted mask and it is equal to one, otherwise.

Mean Intersection over Union is the IoU over all classes in all images:

$$MIoU = \frac{1}{c} \sum_{i} \frac{n_{ii}}{\sum_{j=1}^{c} n_{ij} + \sum_{i} n_{ji} - n_{ii}} \tag{3.7}$$

Equations 3.4, 3.5, and 3.7 are taken from [4]. Equations 3.1, 3.2, 3.3 and 3.6 are taken from [1].

3.4 Implementation

For the computing of the CNN-kernel we used the implementation of Adrià Garriga-Alonso [6]. For data pre-processing we used Numpy [8], pyTorch library [17], and h5py [3]. We used the linear solver from SciPy library [29]. To obtain the patches from large images we used the Patchify library [18]. In addition, to evaluate the model we used the scikit-learn [19]. The ground truth masks predicted segmentation maps are created with the help of matplotlib [27].

The runtime is presented in table 3.2, to compute the kernel matrix between more than five images of size $[256 \times 256]$ using the local computer, with properties shown in table 3.1, we get an out-of-memory error. Hence, we computed the kernel matrix between data samples on the GPU available on *atsccs68* compute node in Garching.

Table 3.1: Technical properties of the local machine, which we used to compute the two operations of solving the linear system and make the predictions, and the atsccs68 node, which we used to compute the kernel matrix.

	local machine	atsccs68-machine
CPU	i7-8550U @2.00 GHz	E5-2670 @2.60GHz
GPU	no GPU is available	GTX 770
OS	Windows 10	Ubuntu 20.04.2 LTS
Kernel	-	Linux 5.4.0-81-generic x86_64
RAM	8.00 GB	16 GB

Table 3.2: Computation time in seconds for each operation and indication, whether the operation is computed on The GPU or the CPU of the local machine.

	GPU	CPU	Computation time
Kernel matrix	x		31.04 s
Solve linear system		x	15.2 s
Make prediction (matrix multiplication)		x	10.01 s

3.5 Performance Evaluation

In this section, we show the evaluation and testing set results and provide the used CNN-kernel architecture. First, we present the performance on the MNIST data set and then on the microscopy images data set. We optimized the hyperparameter: number of layers, kernel size, bias σ_b^2, weight σ_w^2, the padding of the convolution and the activation function. We compute every time the evaluation metrics on the validation data set. Once we pick the best hyperparameter we evaluate the model on the testing data set.

3.5.1 MNIST Results

The MNIST data set [14] is usually used for classification tasks. However, we performed semantic segmentation. Instead of assigning a label to the whole image, we assigned a label (0 for the background and 1 for the digit itself) to each image pixel individually.

The data set contains 28x28 pixels gray-scale images of handwritten digitals ranging from 0 to 9, 60000 samples for training, and 10000 samples for testing. We used a subset of the MNIST dataset, where data samples are selected randomly. The subsets are balanced and disjoint.

Table 3.3 below shows the number of data samples in each training, validation, and testing subsets. Table 3.4 shows the used hyperparameter of the ConvNet GP. After training the GP classifier on the validation and testing set, we computed the evaluation metrics. Results are shown in tables 3.5 and 3.6. Finally, some of the predicted masks for images from the testing set are illustrated in figure 3.5. Our model performed extremely well, with nearly 97% for mean accuracy and 95.5% for mean IoU on the validation and testing set. Thus we showed the potential of GP-CNN kernels for the pixel-wise prediction task.

Table 3.3: MNIST data split.

	Training	Validation	Testing
# Samples	500	100	100

Table 3.4: Hyperparameter of ConvNet GP.

Hyperparameter	**Value**
Number of layers	7
Kernel size	7
σ_w^2	2.79
σ_b^2	7.86
Padding	SAME
Activation function	RELU

Table 3.5: Performance on the MNIST data set on the **validation** set: precision, recall, F1-score and intersection over mean is calculated for each label. The average IoU and accuracy are calculated. Results are in percent (%).

	Precision	Recall	F1-score	IoU	Mean Accuracy	Mean IoU
Background	97.8	98.6	98.2	96.5	97.1	94.5
Digit	93.89	90.7	92.2	85.6		

Table 3.6: Performance on the MNIST data set on the **Testing** set: precision, recall, F1-score and intersection over mean is calculated for each label. The average IoU and accuracy are calculated. Results are in percent (%).

	Precision	Recall	F1-score	IoU	Mean Accuracy	Mean IoU
Background	97.8	98.8	98.3	96.6	97.3	95.5
Digit	94.5	91.0	92.6	86.3		

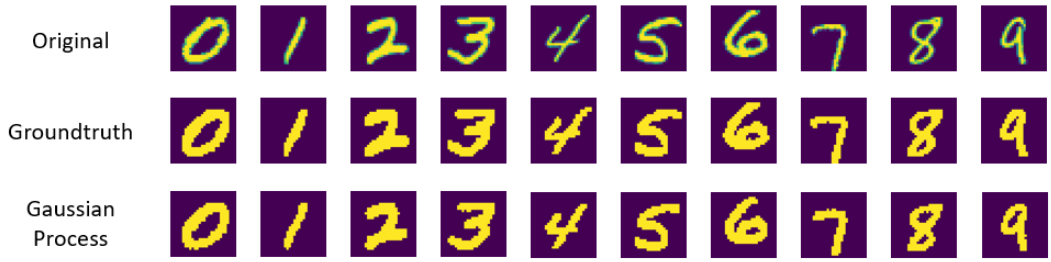


Figure 3.5: Prediction of Gaussian process classifier model, compared to the ground-truth mask and the corresponding gray-scale image from the **testing** set.

3.5.2 Microscopy Data Set Results

This subsection contains the performance results on the validation and the testing data sets. Table 3.7 demonstrates the hyperparameter of the ConvNet-GP. The neural network consists of two convolutional layers, each uses filter with kernel size 3×3. Each convolutional layer is followed by ReLU activation. The final layer uses a 256×256 filter with no padding, which equivalent to a dense layer.

Table 3.7: Hyperparameter of ConvNet GP.

Hyperparameter	**Value**
Number of layers	2
Kernel size	3
σ_w^2	2.79
σ_b^2	7.86
Padding	SAME
Activation function	RELU

We will use three different data sets in order first to find out which step size is better than the other, secondly to see if more data samples in the training subset affect the model performance:

- **MI_128_S**: contains 50 samples in each subset training, validation and testing.
- **MI_128_E**: contains 425, 76, and 58 samples on each training, validation, and testing subset. In addition, 128 refers to the step size when cropping the image into patches.

- **MI_64_E**: contains 599, 74, and 74 samples on each training, validation, and testing subset. In addition, 64 refers to the step size when cropping the image into patches.

Additionally, we will train two models:

- **4-classes model**: we have four objects to predict the background, hyphae, arbuscules, and vesicles.
- **2-classes model**: The model is trained to predict two objects background and arbuscules+vesicles. Therefore, we omitted hyphae by merging them with the background, and we consider arbuscules and vesicles as the same class label.

Likening table 3.8 with table 3.10, we observe that adding more training samples increases the overall model performance by 11.6% for the mean IoU and by 15.95% for the mean F1-score on the validation set. Further, when comparing table 3.10 with table 3.14 and table 3.11 with table 3.15. On the one hand, we conclude that the 128 step size patches data set slightly outperforms the 64 step size patches data set using the 4-classes model on validation and testing sets. For the 128 step size, the mean IoU and the mean F1-score increased by 6.3% on the validation set. The mean IoU and the mean F1-score on the testing set increased by 8.4% and 12.35%, respectively. On the other hand, when using the 2-classes model, we do not notice any remarkable improvement; the results look quite the same for 128 step size and 64 step size.

Moreover, according to Tables 3.11 and 3.13, we can compare the performance of the validation and testing sets. Using the 4-classes model, the mean IoU and F1-score narrowly decrease by 6.5% and 7.5%, respectively. Using the 2-classes model, the IoU and F1-score for the arbusclues and vesicles which belong to the same class label dropped by $5,1\%$. The reason is obviously that the difference between testing set images and training set images is greater than the difference between validation set images and training set.

To sum up, table 3.9 shows that the predictions of the training set are made perfectly, with 100% mean IoU and mean F1-score. Besides, the two models produced unsatisfying predictions on the three different data sets: for the 4-classes model, the mean IoU never exceeded 21.3% for the validation set and 15% for the testing set. Using the 2-classes model, predicting arbuscules and vesicles slightly improved—however, the F1-score only ranged between 26% and 40%. Likewise, based on the F1-score value, arbuscules are marginally predicted correctly compared to hyphae and vesicles. The lower contrast of hyphae regarding arbuscules and the resemblance between arbuscules and vesicles might be the causes for this behavior.

MI_128_S

Table 3.8: 4-classes gaussian process classifier on the **validation** set: precision, recall, F1-score and intersection over mean is calculated for each label. The average IoU and F1-score are calculated without Background. Results are in percent (%).

	Precision	Recall	F1-score	IoU	Mean IoU	Mean F1-score
Background	83.2	93.1	87.8	78.8	9.7	15.51
Hyphae	23.5	15.5	17.59	10.8		
Arbuscules	21.3	10.0	12.5	7.8		
Vesicles	11.0	11.0	9.5	9.5		

MI_128_E

Table 3.9: 4-classes gaussian process classifier on the **training** set: precision, recall, F1-score and intersection over mean is calculated for each label. The average IoU and F1-score are calculated without Background. Results are in percent (%).

	Precision	Recall	F1-score	IoU	Mean IoU	Mean F1-score
Background	100.0	100.0	100.0	100.0	100.0	100.0
Hyphae	100.0	100.0	100.0	100.0		
Arbuscules	100.0	100.0	100.0	100.0		
Vesicles	100.0	100.0	100.0	100.0		

Table 3.10: 4-classes gaussian process classifier on the **validation** set: precision, recall, F1-score and intersection over mean is calculated for each label. The average IoU and F1-score are calculated without Background. Results are in percent (%).

	Precision	Recall	F1-score	IoU	Mean IoU	Mean F1-score
Background	86.1	95.5	90.5	83.0	21.3	31.46
Hyphae	38.9	26.4	30.4	20.0		
Arbuscules	51.6	27.8	33.7	23.0		
Vesicles	45.8	20.7	25.1	18.3		

Table 3.11: 4-classes gaussian process classifier on the **testing** set: precision, recall, F1-score and intersection over mean is calculated for each label. The average IoU and F1-score are calculated without Background. Results are in percent (%).

	Precision	Recall	F1-score	IoU	Mean IoU	Mean F1-score
Background	85.7	95.19	90.1	82.4	14.8	23.92
Hyphae	31.0	19.6	23.4	14.0		
Arbuscules	40.3	20.3	26.0	16.2		
Vesicles	30.8	11.3	14.0	9.3		

Table 3.12: 2-classes gaussian process classifier on the **validation** set: precision, recall, F1-score and intersection over mean is calculated for each label. Results are in percent (%).

	Precision	Recall	F1-score	IoU
Background	93.5	98.5	95.89	92.2
Arbuscules + Vesicles	63.1	29.4	37.5	25.6

Table 3.13: 2-classes gaussian process classifier on the **testing** set: precision, recall, F1-score and intersection over mean is calculated for each label. Results are in percent (%).

	Precision	Recall	F1-score	IoU
Background	93.1	98.4	95.7	91.8
Arbuscules + Vesicles	61.6	23.3	32.3	20.5

MI_64_E

Table 3.14: 4-classes gaussian process classifier on the **validation** set: precision, recall, F1-score and intersection over mean is calculated for each label. The average IoU and F1-score are calculated without Background. Results are in percent (%).

	Precision	Recall	F1-score	IoU	Mean IoU	Mean F1-score
Background	84.1	94.5	89.0	80.3	15.0	25.01
Hyphae	27.7	16.6	20.3	11.5		
Arbuscules	48.9	27.8	34.9	21.9		
Vesicles	48.9	10.5	15.9	9.5		

Table 3.15: 4-classes gaussian process classifier on the **testing** set: precision, recall, F1-score and intersection over mean is calculated for each label. The average IoU and F1-score are calculated without Background. Results are in percent (%).

	Precision	Recall	F1-score	IoU	Mean IoU	Mean F1-score
Background	79.8	87.2	83.2	71.7	6.4	11.57
Hyphae	13.7	8.1	10.0	5.4		
Arbuscules	19.2	12.0	14.39	8.2		
Vesicles	9.7	8.4	8.1	4.5		

Table 3.16: 2-classes gaussian process classifier on the **validation** set: precision, recall, F1-score and intersection over mean is calculated for each label. Results are in percent (%).

	Precision	Recall	F1-score	IoU
Background	92.8	98.2	95.39	92.2
Arbuscules + Vesicles	63.8	29.9	40.3	25.6

Table 3.17: 2-classes gaussian process classifier on the **testing** set: precision, recall, F1-score and intersection over mean is calculated for each label. Results are in percent (%).

	Precision	Recall	F1-score	IoU
Background	92.2	91.6	91.9	85.2
Arbuscules + Vesicles	27.0	26.2	26.1	15.4

3.5.3 Visual results

In this section we illustrate a few predictions of the Gaussian process classifier compared with the ground-truth segmentation map for the training, validation, and testing set. In the below figures, on the right, we show the real image, on the middle, the ground-truth, and on the left, from the Gaussian process classifier predicted segmentation map. Figures 3.6, 3.7, 3.8, 3.9, 3.12, and 3.13 hyphae are shown in red, arbuscules in yellow, and vesicles in blue. Figures 3.10, 3.11, 3.14, and 3.15 vesicles and arbuscules are shown in blue.

Figure 3.6 demonstrates four prediction segmentation maps from the validation set using the small training set of only 50 data samples. The GP makes a satisfying prediction on images, which are pretty similar to some samples in the training set. We can see an example of it in the first image of the figure. In addition, sometimes, the GP invented stuff, as shown in the fourth image of the figure.

Figure 3.7 shows four prediction segmentation maps from the extended training set [MI_128_E]. As illustrated, the predicted segmentation mask is identical to the provided ground truth of the real images. While figures 3.8 and 3.9 demonstrate four prediction segmentation maps from the validation and testing set, respectively. The predictions are very noisy, where the model scarcely recognizes vesicles. Figures 3.10 and 3.11 show four prediction segmentation maps of the 2-classes model from the validation and testing set. Where we again notice that the prediction are imprecise.

For the data set, where the large image being cropped into patches with step size of 64, figure 3.12 shows that the predicted segmentation maps of the validation set are again noisy and inexact. Vesicles are most of the time predicted as background and arbuscules are sometimes predicted as vesicles. According to figure 3.13, the model is coming up with other predictions that do not match the ground truth. Whereas Figure 3.14 demonstrates the predicted segmentation maps of four samples from the validation, and Figure 3.15 from the testing set, using the 2-classes model.

MI_128_S

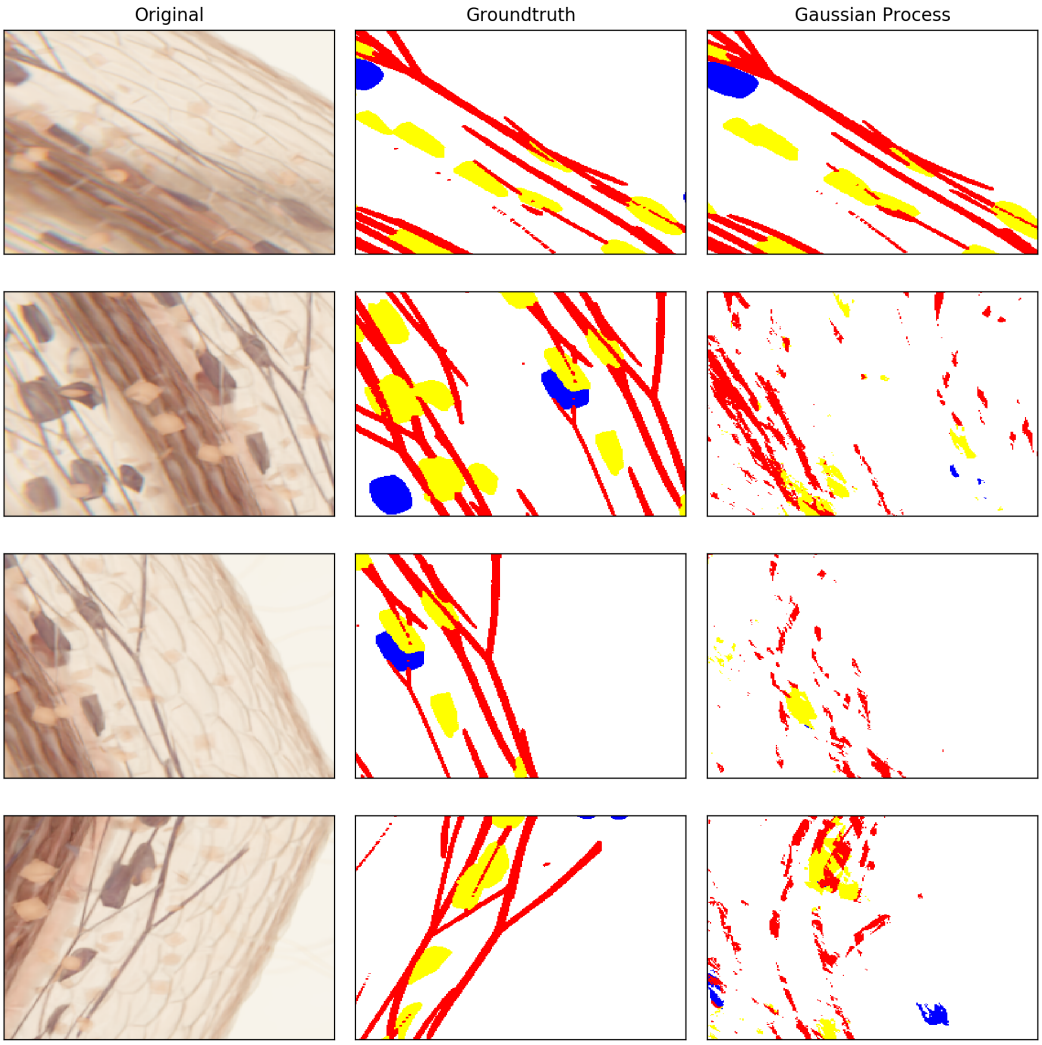


Figure 3.6: Predictions of the 4-classes based Gaussian process model, compared to the ground-truth and the corresponding rendered image from the **validation** set.

MI_128_E

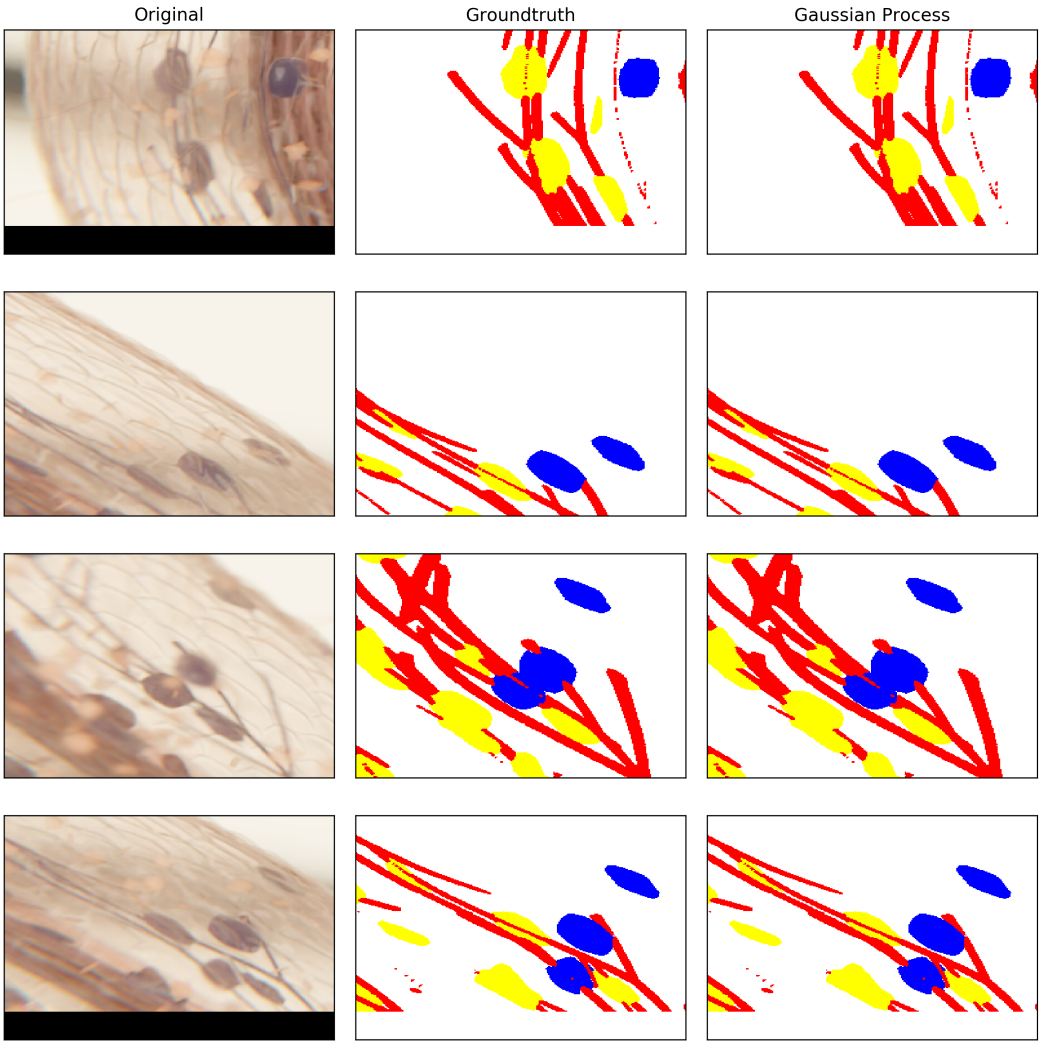


Figure 3.7: Predictions of the 4-classes based Gaussian process model, compared to the ground-truth and the corresponding rendered image from the **training** set.

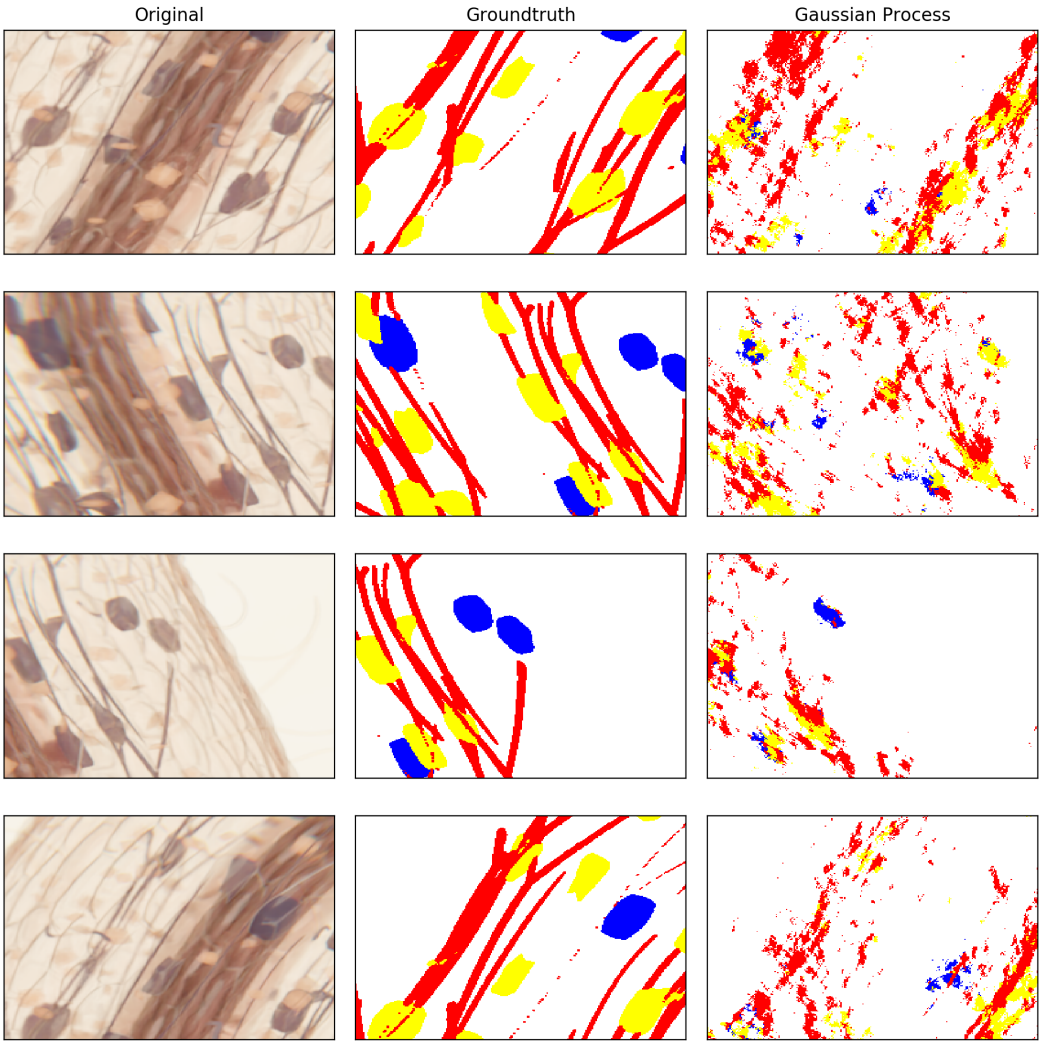


Figure 3.8: Predictions of the 4-classes based Gaussian process model, compared to the ground-truth and the corresponding rendered image from the **validation** set.

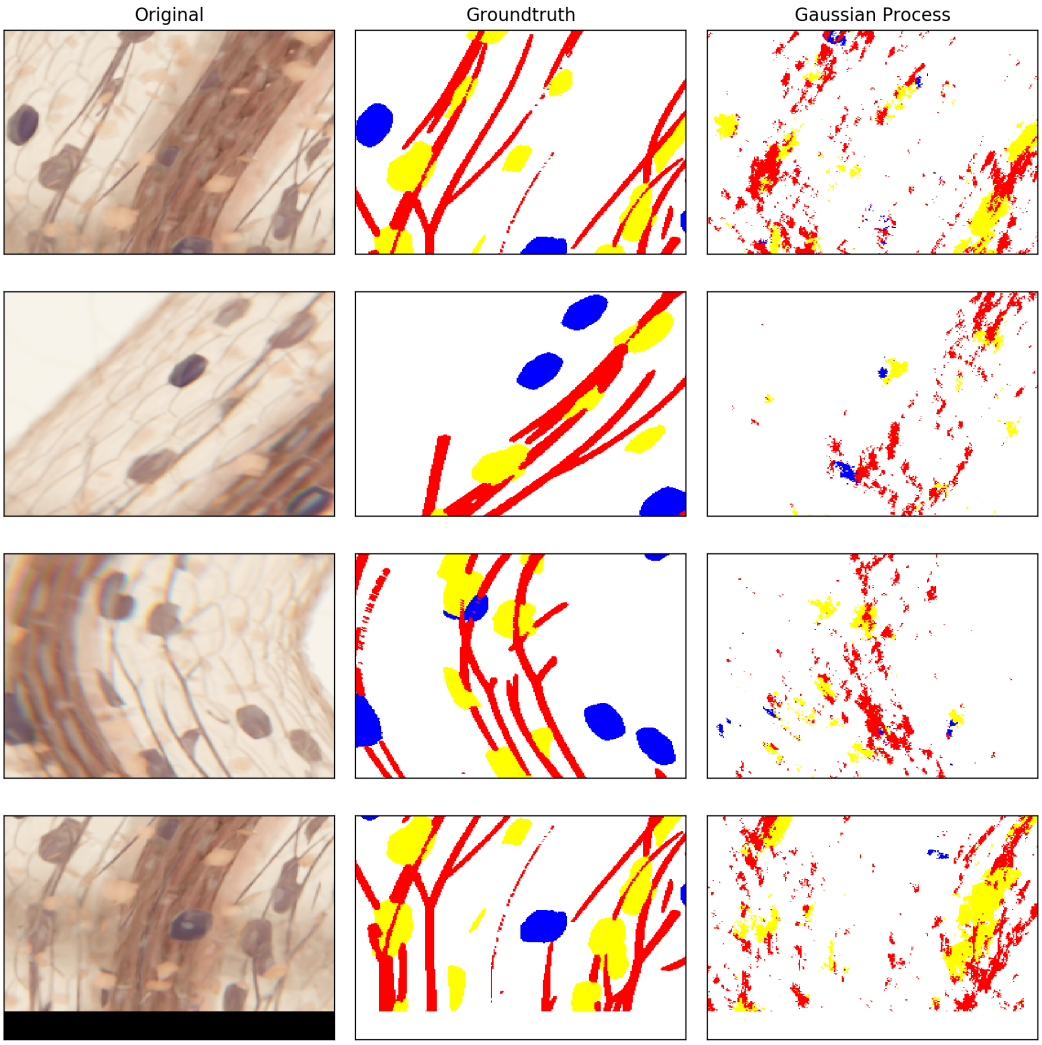


Figure 3.9: Predictions of the 4-classes based Gaussian process model, compared to the ground-truth and the corresponding rendered image from the **testing** set.

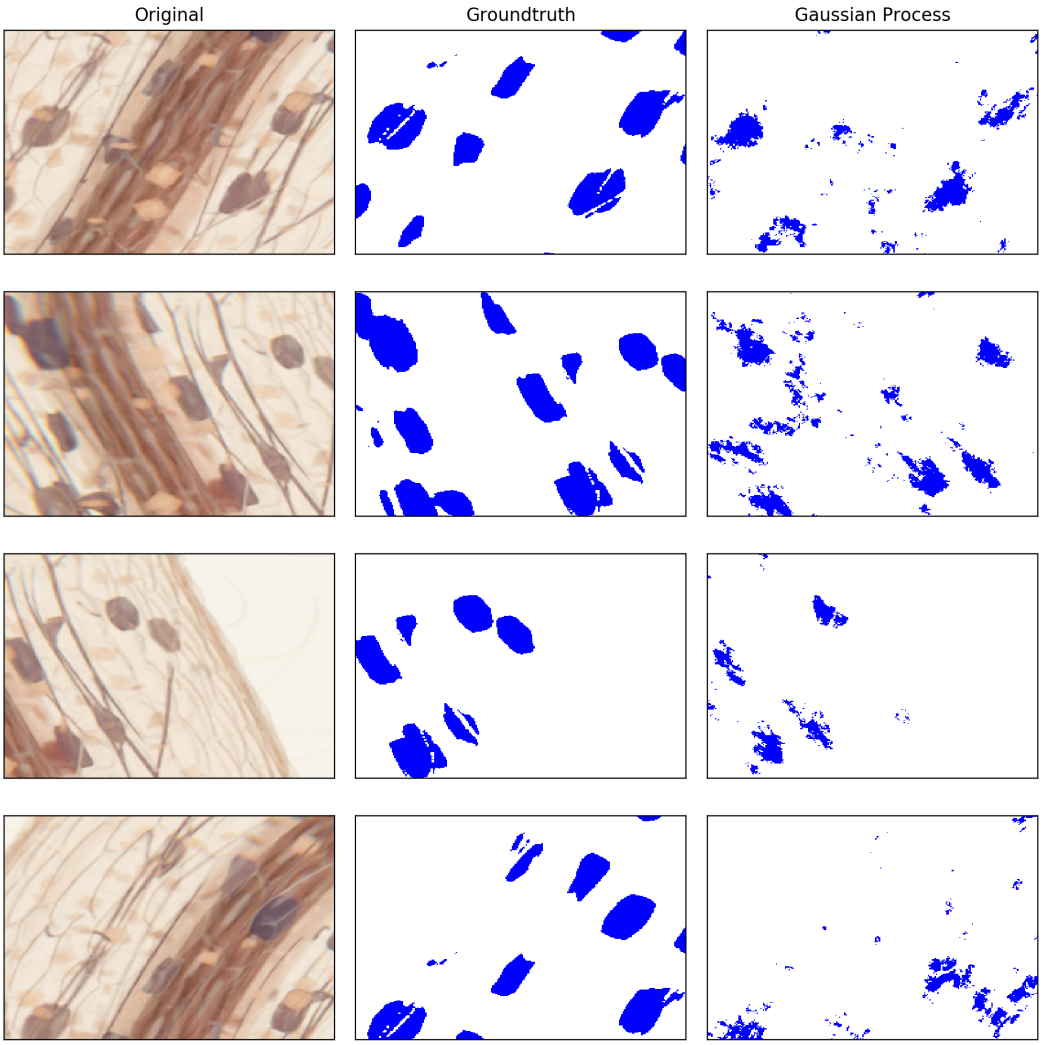


Figure 3.10: Predictions of the 2-classes based Gaussian process model, compared to the ground-truth and the corresponding rendered image from the **validation** set.

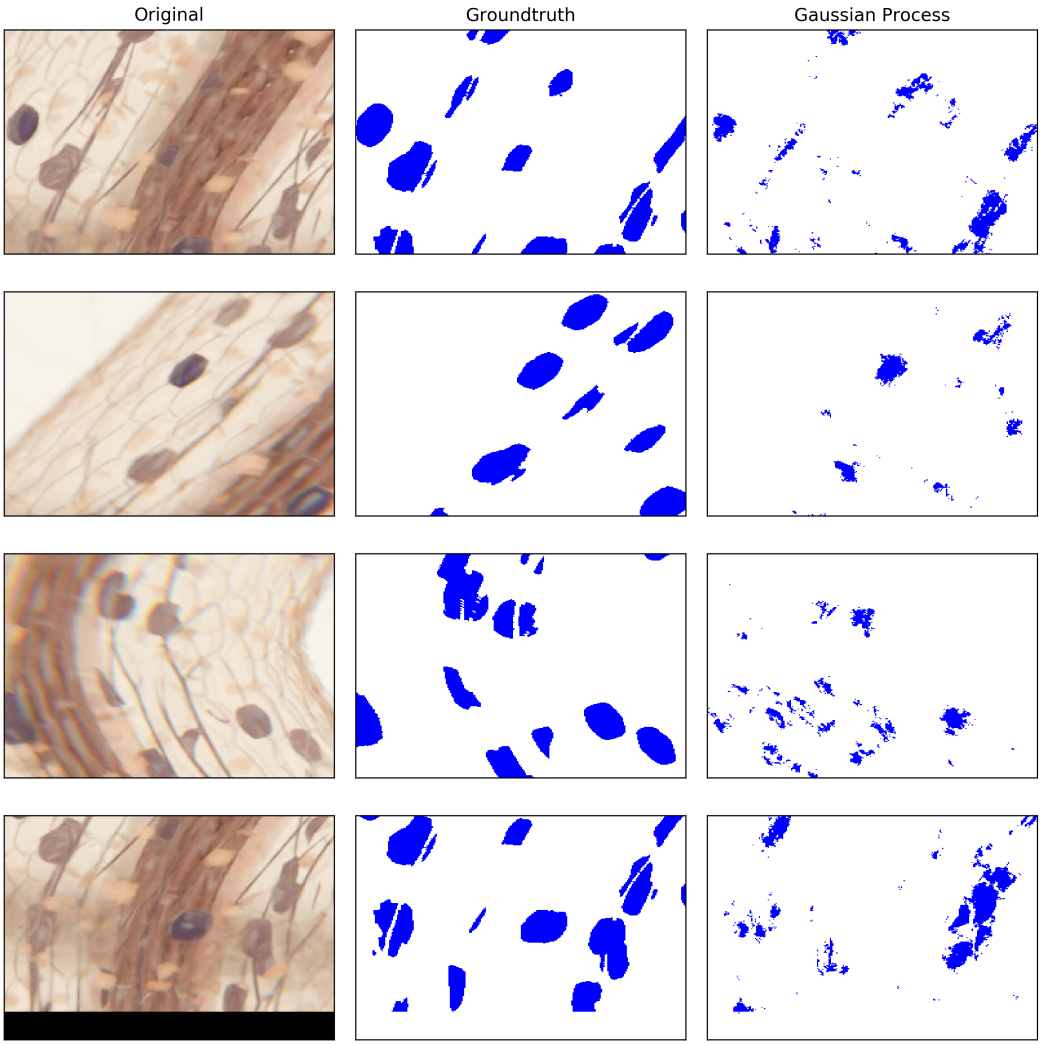


Figure 3.11: Predictions of the 2-classes based Gaussian process model, compared to the ground-truth and the corresponding rendered image from the **testing** set.

MI_64_E

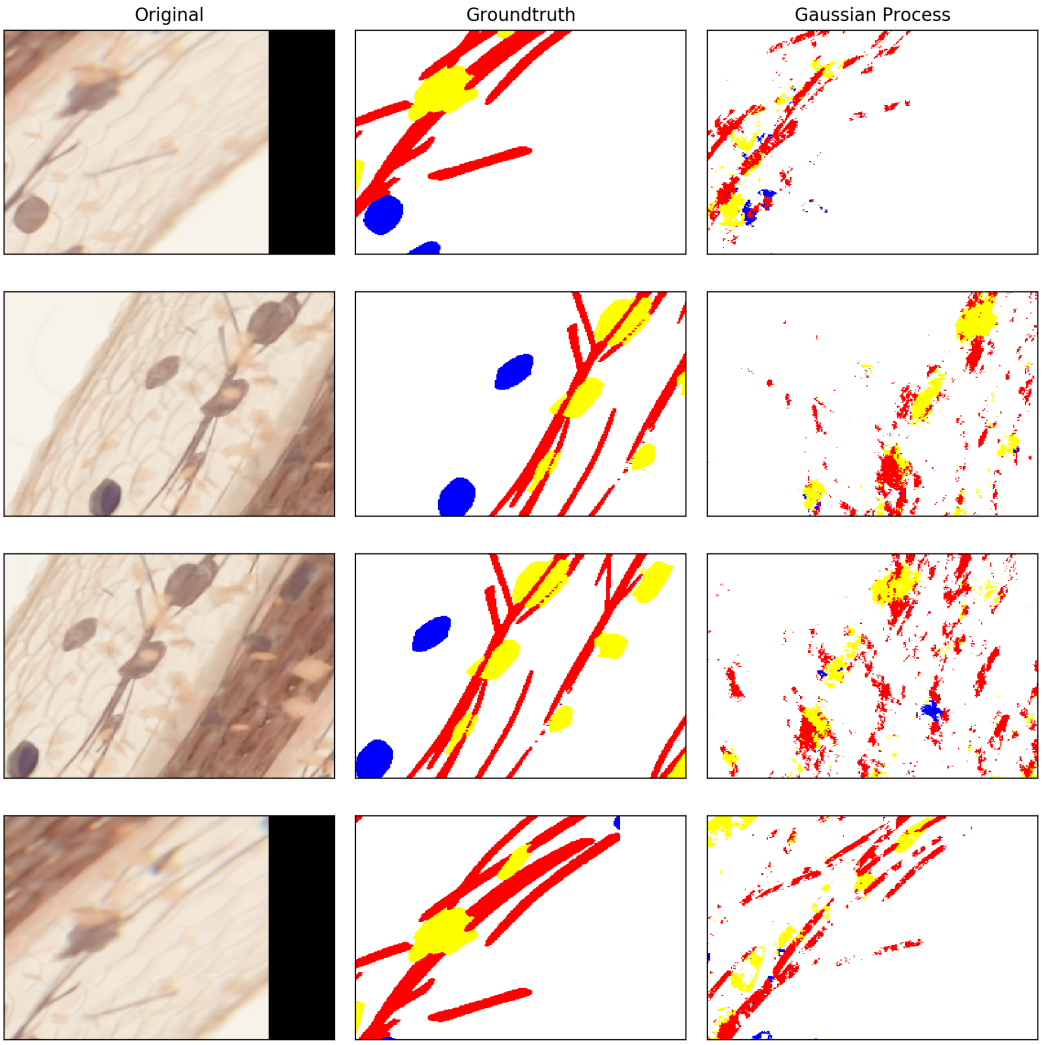


Figure 3.12: Predictions of the 4-classes based Gaussian process model, compared to the ground-truth and the corresponding rendered image from the **validation** set.

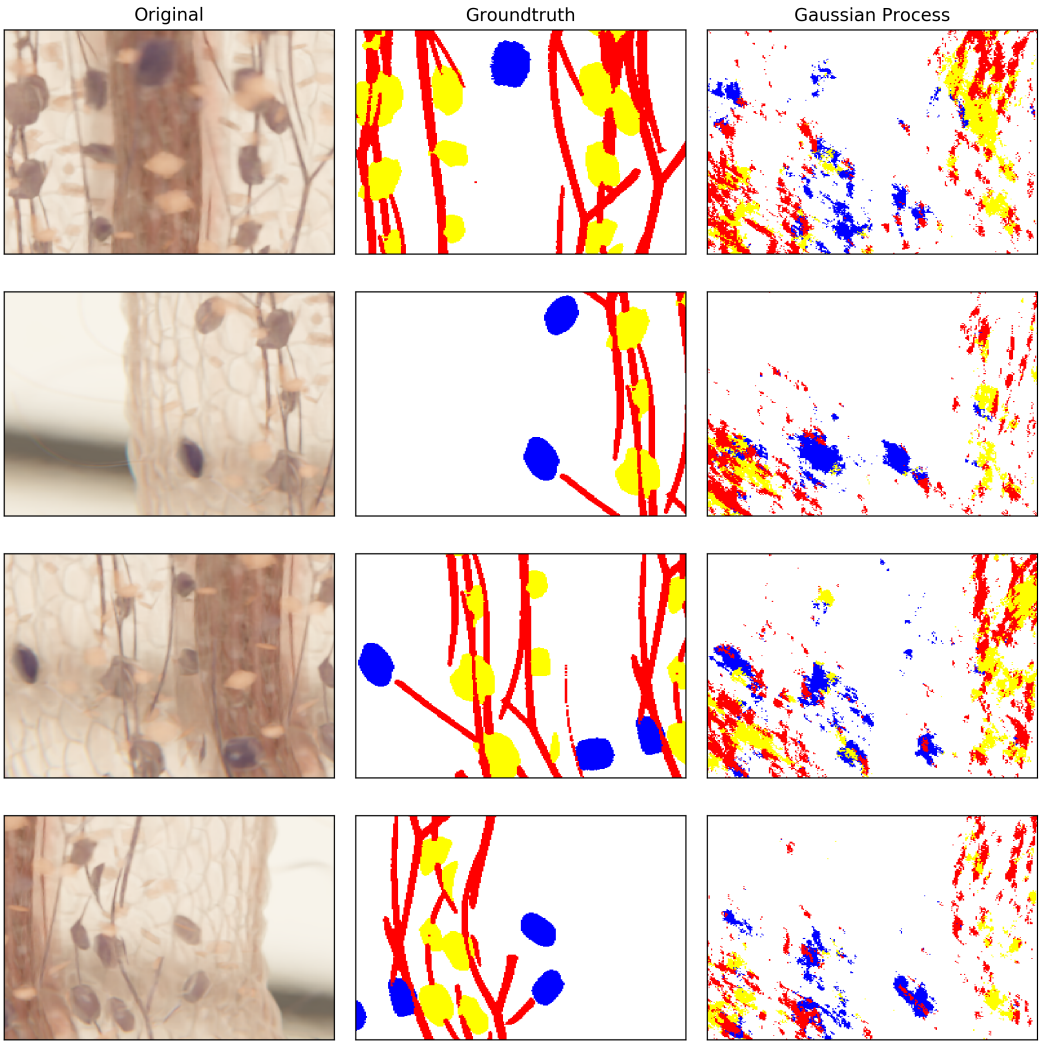


Figure 3.13: Predictions of the 4-classes based Gaussian process model, compared to the ground-truth and the corresponding rendered image from the **testing** set.

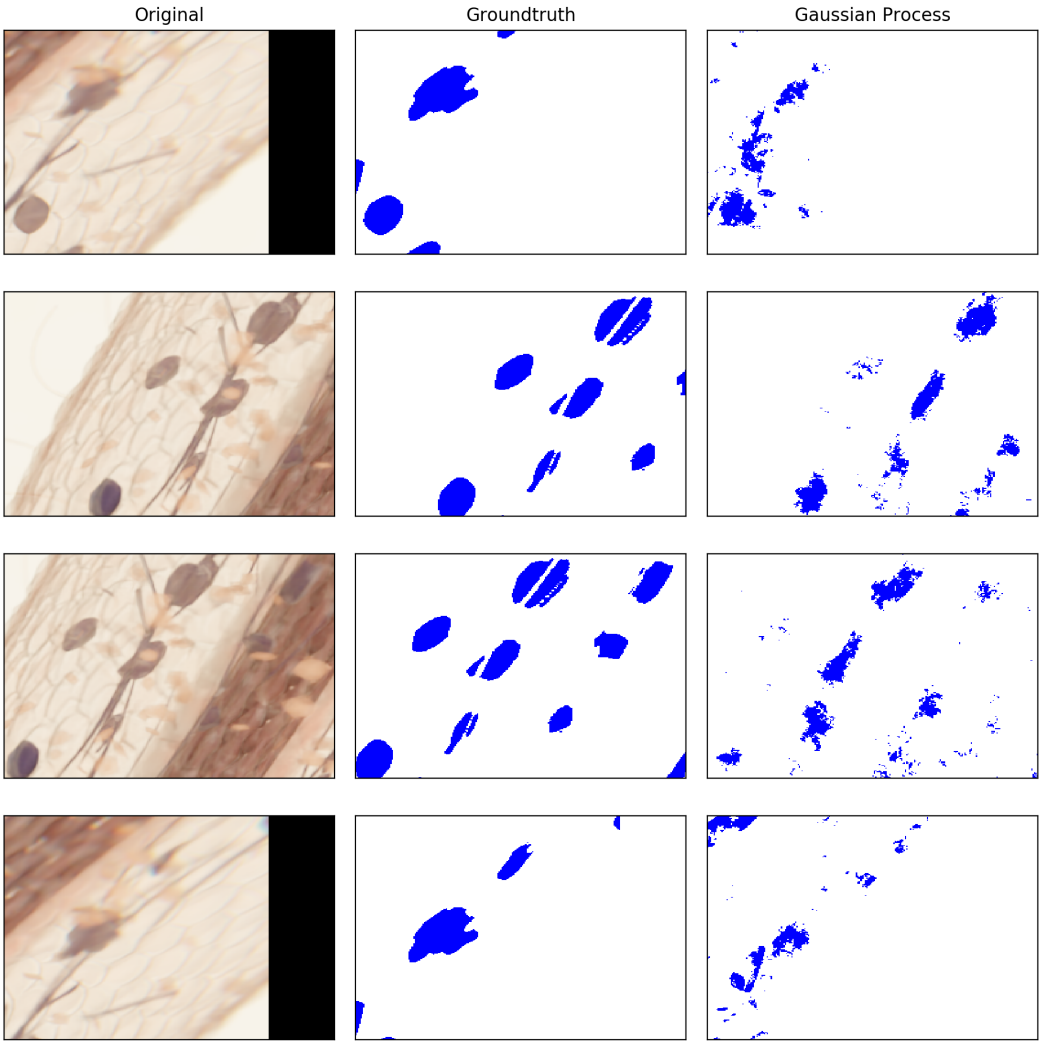


Figure 3.14: Predictions of the 2-classes based Gaussian process model, compared to the ground-truth and the corresponding rendered image from the **validation** set.

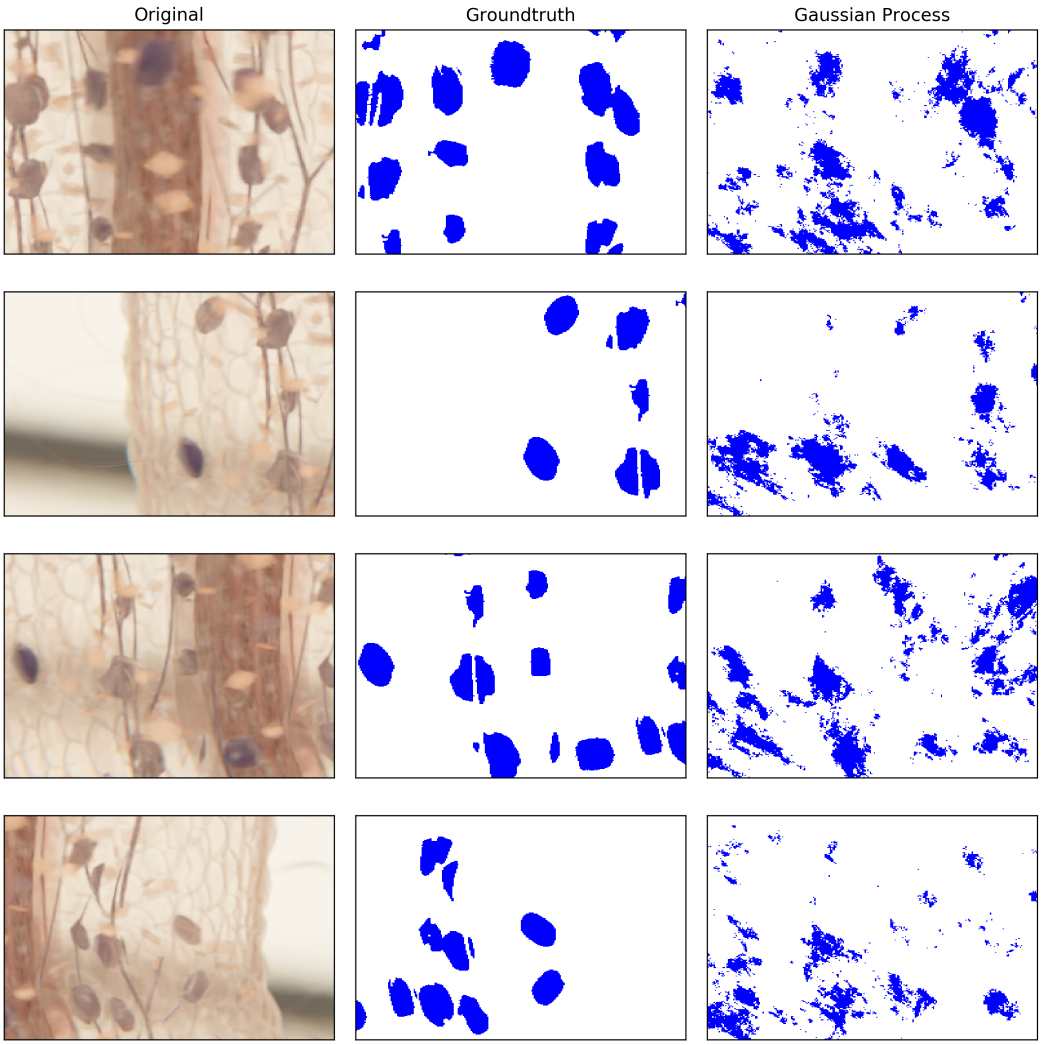


Figure 3.15: Predictions of the 2-classes based Gaussian process model, compared to the ground-truth and the corresponding rendered image from the **testing** set.

4 Conclusion

In this thesis, we presented how to apply Bayesian classification using Gaussian processes to the challenging task of semantic segmentation of light microscopy images. Before that, we explained Gaussian processes and their connection to convolutional neural networks, how they could be used for regression and classification, and provided a short overview of deep learning techniques for solving image segmentation tasks.

We presented the quantitative and visual results to evaluate our approach. We can clearly see that, on the one hand, the Gaussian process classifier did well on data samples that are very similar to training samples. However, the predictions of unseen data samples are scattered on the other hand. In fact, the GPC has problems with granularity. As explained in section 2.5.2, the encoder-decoder-based U-Net model achieved outstanding results for semantic segmentation of microscopy images. Multiple upsampling layers and a skip connection technique were deployed to combine semantic and spatial information to get smoother segmentation results. That can explain our approach's behavior that does not have an encoder part. Another cause of this behavior could be that we did not find the right choice of hyperparameter value.

In our work, we did not take advantage of Gaussian processes uncertainty estimation. Therefore, we can extend the approach for future work by incorporating it in an encoder-decoder architecture as done by Joakim Johnander et al [10]. Furthermore, we can automate hyperparameter optimization to improve the model's performance.

List of Figures

2.1 AMF structure, taken from [30]. 4
2.2 Annotated colonized root section. Hyphae are annotated with yellow lines, arbusclues with red circles, and vesicles with green circles, taken from [30]. 5
2.3 Render image on the left, segmentation map on the right. Vesicles (blue), arbuscules (yellow) and hyphae (red). 6
2.4 2D Convolutional, where the kernel $U_{i,j}^{(0)}$ is applied to an input image x_j, taken from [6]. 11
2.5 Example of a semantic, instance and panoptic segmentation. Taken from [12]. 14
2.6 A Fully Connected Network that learns pixel-wise predictions, taken from [15]. 15
2.7 SegNet; The encoder applies convolutions and max pooling. The decoder up-samples the encoder output using the max pooling indices. Finally the softmax layer performs pixel-wise prediction. Taken from [1]. 16

3.1 Illustration of the steps for creating the patches data set. 18
3.2 Split the image into overlapping patches. The blue, red, and green blocks show an example of three overlapping patches on the left—some patch samples on the right. 18
3.3 Flow diagram of pre-processing steps: the right side shows the transformations for the rendered image, the left side shows the transformations for the segmentation maps. 19
3.4 Overview of our approach: the training images, corresponding segmentation maps, and the unseen images set are first pre-processed and then fed into the Gaussian Process Classifier. Finally, we get the segmentation masks Y^* as output. 20
3.5 Prediction of Gaussian process classifier model, compared to the ground-truth mask and the corresponding gray-scale image from the **testing** set. 25

3.6 Predictions of the 4-classes based Gaussian process model, compared to the ground-truth and the corresponding rendered image from the **validation** set. . . . 31
3.7 Predictions of the 4-classes based Gaussian process model, compared to the ground-truth and the corresponding rendered image from the **training** set. . . . 32
3.8 Predictions of the 4-classes based Gaussian process model, compared to the ground-truth and the corresponding rendered image from the **validation** set. . . . 33
3.9 Predictions of the 4-classes based Gaussian process model, compared to the ground-truth and the corresponding rendered image from the **testing** set. . . . 34
3.10 Predictions of the 2-classes based Gaussian process model, compared to the ground-truth and the corresponding rendered image from the **validation** set. . . . 35
3.11 Predictions of the 2-classes based Gaussian process model, compared to the ground-truth and the corresponding rendered image from the **testing** set. . . . 36
3.12 Predictions of the 4-classes based Gaussian process model, compared to the ground-truth and the corresponding rendered image from the **validation** set. . . . 37
3.13 Predictions of the 4-classes based Gaussian process model, compared to the ground-truth and the corresponding rendered image from the **testing** set. . . . 38
3.14 Predictions of the 2-classes based Gaussian process model, compared to the ground-truth and the corresponding rendered image from the **validation** set. . . . 39
3.15 Predictions of the 2-classes based Gaussian process model, compared to the ground-truth and the corresponding rendered image from the **testing** set. . . . 40

List of Tables

3.1 Technical properties of the local machine, which we used to compute the two operations of solving the linear system and make the predictions, and the atsccs68 node, which we used to compute the kernel matrix. . . 22
3.2 Computation time in seconds for each operation and indication, whether the operation is computed on The GPU or the CPU of the local machine. 23
3.3 MNIST data split. 24
3.4 Hyperparameter of ConvNet GP. 24
3.5 Performance on the MNIST data set on the **validation** set: precision, recall, F1-score and intersection over mean is calculated for each label. The average IoU and accuracy are calculated. Results are in percent (%). 24
3.6 Performance on the MNIST data set on the **Testing** set: precision, recall, F1-score and intersection over mean is calculated for each label. The average IoU and accuracy are calculated. Results are in percent (%). . . 24
3.7 Hyperparameter of ConvNet GP. 25
3.8 4-classes gaussian process classifier on the **validation** set: precision, recall, F1-score and intersection over mean is calculated for each label. The average IoU and F1-score are calculated without Background. Results are in percent (%). 27
3.9 4-classes gaussian process classifier on the **training** set: precision, recall, F1-score and intersection over mean is calculated for each label. The average IoU and F1-score are calculated without Background. Results are in percent (%). 27
3.10 4-classes gaussian process classifier on the **validation** set: precision, recall, F1-score and intersection over mean is calculated for each label. The average IoU and F1-score are calculated without Background. Results are in percent (%). 27
3.11 4-classes gaussian process classifier on the **testing** set: precision, recall, F1-score and intersection over mean is calculated for each label. The average IoU and F1-score are calculated without Background. Results are in percent (%). 28

3.12 2-classes gaussian process classifier on the **validation** set: precision, recall, F1-score and intersection over mean is calculated for each label. Results are in percent (%). 28
3.13 2-classes gaussian process classifier on the **testing** set: precision, recall, F1-score and intersection over mean is calculated for each label. Results are in percent (%). 28
3.14 4-classes gaussian process classifier on the **validation** set: precision, recall, F1-score and intersection over mean is calculated for each label. The average IoU and F1-score are calculated without Background. Results are in percent (%). 29
3.15 4-classes gaussian process classifier on the **testing** set: precision, recall, F1-score and intersection over mean is calculated for each label. The average IoU and F1-score are calculated without Background. Results are in percent (%). 29
3.16 2-classes gaussian process classifier on the **validation** set: precision, recall, F1-score and intersection over mean is calculated for each label. Results are in percent (%). 29
3.17 2-classes gaussian process classifier on the **testing** set: precision, recall, F1-score and intersection over mean is calculated for each label. Results are in percent (%). 29

Bibliography

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation." In: *IEEE transactions on pattern analysis and machine intelligence* 39.12 (2017), pp. 2481–2495.

[2] Juan C Caicedo et al. "Evaluation of deep learning strategies for nucleus segmentation in fluorescence images." In: *Cytometry Part A* 95.9 (2019), pp. 952–965.

[3] Andrew Collette et al. "H5Py/H5Py: 2.4. 0." In: (2017).

[4] Soumyabrata Dev et al. "Localizing adverts in outdoor scenes." In: *2019 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*. IEEE. 2019, pp. 591–594.

[5] Chuong B Do. "The multivariate Gaussian distribution." In: *Section Notes, Lecture on Machine Learning, CS* 229 (2008).

[6] Adrià Garriga-Alonso, Carl Edward Rasmussen, and Laurence Aitchison. "Deep convolutional networks as shallow gaussian processes." In: *arXiv preprint arXiv:1808.05587* (2018).

[7] Anirudha Ghosh et al. "Fundamental concepts of convolutional neural network." In: *Recent Trends and Advances in Artificial Intelligence and Internet of Things*. Springer, 2020, pp. 519–567.

[8] Charles R Harris et al. "Array programming with NumPy." In: *Nature* 585.7825 (2020), pp. 357–362.

[9] Achim Hekler et al. "Deep learning outperformed 11 pathologists in the classification of histopathological melanoma images." In: *European Journal of Cancer* 118 (2019), pp. 91–96.

[10] Joakim Johnander et al. "Dense Gaussian Processes for Few-Shot Segmentation." In: *arXiv preprint arXiv:2110.03674* (2021).

[11] Li-Hong Juang and Ming-Ni Wu. "MRI brain lesion image detection based on color-converted K-means clustering segmentation." In: *Measurement* 43.7 (2010), pp. 941–949.

[12] Alexander Kirillov et al. "Panoptic segmentation." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 9404–9413.

[13] Nikolaus Korfhage et al. "Detection and segmentation of morphologically complex eukaryotic cells in fluorescence microscopy images via feature pyramid fusion." In: *PLOS Computational Biology* 16.9 (2020), e1008179.

[14] Yann LeCun, Corinna Cortes, and Chris Burges. *MNIST handwritten digit database*. 2010.

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

[16] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[17] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library." In: *Advances in neural information processing systems* 32 (2019).

[18] *patchify python library*. `https://pypi.org/project/patchify/`. Accessed: 2022-04-30.

[19] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python." In: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830.

[20] Sebastian G Popescu et al. "Distributional gaussian process layers for outlier detection in image segmentation." In: *International Conference on Information Processing in Medical Imaging*. Springer. 2021, pp. 415–427.

[21] Carl Edward Rasmussen. "Gaussian processes in machine learning." In: *Summer school on machine learning*. Springer. 2003, pp. 63–71.

[22] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[23] Torgyn Shaikhina and Natalia A Khovanova. "Handling limited datasets with neural networks in medical applications: A small-data approach." In: *Artificial intelligence in medicine* 75 (2017), pp. 51–63.

[24] Sally E Smith and David J Read. *Mycorrhizal symbiosis*. Academic press, 2010.

[25] Marina Sokolova, Nathalie Japkowicz, and Stan Szpakowicz. "Beyond accuracy, F-score and ROC: a family of discriminant measures for performance evaluation." In: *Australasian joint conference on artificial intelligence*. Springer. 2006, pp. 1015–1021.

[26] Eleftherios Spyromitros-Xioufis et al. "Multi-label classification methods for multi-target regression." In: *arXiv preprint arXiv:1211.6581* (2012), pp. 1159–1168.

[27] Sandro Tosi. *Matplotlib for Python developers*. Packt Publishing Ltd, 2009.

[28] Thanh Tran et al. "Blood cell count using deep learning semantic segmentation." In: (2019).

[29] Pauli Virtanen et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." In: *Nature methods* 17.3 (2020), pp. 261–272.

[30] Jan Watter. "Light Microscopy Image Analysis using Neural Networks." In: (2021).

[31] Teresa Wu et al. "A prior feature SVM-MRF based method for mouse brain segmentation." In: *Neuroimage* 59.3 (2012), pp. 2298–2306.