SCHOOL OF COMPUTATION,
INFORMATION AND TECHNOLOGY

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Gaussian Processes for Light Microscopy Image Segmentation

Paul Ungermann

SCHOOL OF COMPUTATION,
INFORMATION AND TECHNOLOGY

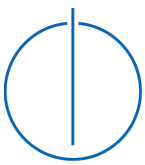
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Gaussian Processes for Light Microscopy Image Segmentation

Gaußprozesse zur Segmentierung von Lichtmikroskopiebildern

Author:	Paul Ungermann
Supervisor:	Dr. Felix Dietrich
Submission Date:	15.03.2023



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

Munich, 15.03.2023 Paul Ungermann

Acknowledgments

Firstly, I want to thank my advisor Dr. Felix Dietrich for the possibility of writing my bachelor's thesis at the Chair of Scientific Computing in Computer Science. Throughout this thesis, he provided continuous guidance, support, and encouragement. I am very grateful for his constructive feedback and his insightful suggestions. Furthermore, I would like to also thank Prof. Erik Rodner for giving me more profound insights into his research paper. Of course, special thanks to my friends who supported me throughout my studies.
Finally, I am deeply indebted to my parents, who gave me the opportunity for my personal development.

Abstract

Image segmentation plays a significant role in many applications of computer vision, for example, computer tomography in medicine or face recognition in video surveillance. Image segmentation is a major ongoing problem in computer vision. It aims to classify an image into parts to better analyze and understand it.
Recently, deep learning models achieved excellent results, but several other models are also achieving good results, e.g., support vector machines, Auto-Encoder networks, or Gaussian processes. Gaussian processes have several advantages over other state-of-the-art models, e.g., incorporating prior information about the data and obtaining uncertainty estimates. In this thesis, we use treed Gaussian processes with a convolutional kernel to segment light microscopy images of Arbuscular Mycorrhiza Fungi. A treed Gaussian process is a combination of decision trees and Gaussian processes. It uses a decision tree to partition the input space and then employs one Gaussian process for each data partition, i.e., leaf node of the decision tree. We first discuss the theoretical properties of decision trees and Gaussian processes and how Gaussian processes with a convolutional kernel are related to neural networks. Then, we introduce the concept of treed Gaussian processes.
We demonstrate that treed Gaussian processes can achieve excellent results on the extended MNIST data set with a mean F1-Score of 0.81 and a mean Jaccard index of 0.70. We also demonstrate that our model achieves a mean F1-Score of 0.55 and a mean Jaccard index of 0.43 on the light microscopy images. We conclude that a treed Gaussian process offers an accurate and scalable alternative to the traditional Gaussian process.

Contents

1. Introduction

Light microscopy is a fundamental part of biology research. It has a variety of different applications. Generally, we are interested in analyzing images generated by the microscope. However, manually analyzing light microscopy images is often very time-consuming and prone to human errors. Automated image segmentation can be used to mitigate these issues in some situations and to highlight the essential regions of the image. Image segmentation is a crucial task in image processing. It segments the image into its main components so that it can be analyzed more deeply. Automating this procedure can be challenging because of the image's intrinsic complex structure and the variability of biological samples.
In recent years deep learning methods, especially convolutional neural networks, have been used to achieve state-of-the-art results. Recently, a direct connection between convolutional neural networks and Gaussian processes with a convolutional kernel was shown [8]. Therefore, using a convolutional Gaussian process should produce excellent results as well. The major disadvantage of Gaussian processes is the cubic complexity of training time in the number of data points, and thus, they are only feasible for a small data set. Hence, we propose using treed Gaussian processes to deal with the complexity issue.
The light microscopy images oriented on the Arbuscular Mycorrhiza Fungi were rendered using Blender [37]. We start with the biological background of these Fungi. Then, we give an overview of the current state-of-the-art image segmentation methods. In chapter 2.3 and 2.4, we describe decision trees and Gaussian processes. Next, we combine these two concepts and define and illustrate a treed Gaussian process. Then, we reframe treed Gaussian processes in the context of image segmentation. The implementation of a treed Gaussian process is described in section 3.4. Here we go through the data-processing, the model training, and the inference. At the end of the section, we outline our implementation optimization and compare the CPU and GPU calculation times using a benchmark. Lastly, we evaluate our model on an extended MNIST data set and on the light microscopy images.

2. Related Work

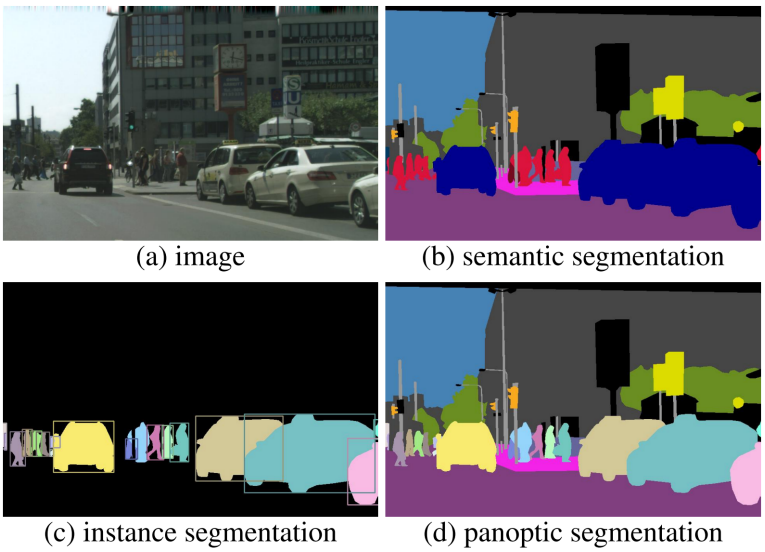
We use treed Gaussian processes with a convolutional kernel to accomplish semantic segmentation on light microscopy images. Firstly, we explain the biological background of Arbuscular Mycorrhiza Fungi, which is used in the provided light microscopy images. Then, we give an overview of current state-of-the-art image segmentation. Next, we explain the mathematical foundations of decision trees, Gaussian processes, and convolutional kernels. Lastly, we introduce treed Gaussian processes.

2.1. Arbuscular Mycorrhiza Fungi

Arbuscular mycorrhiza fungi (AMF) is a soil microorganism that forms a mutualistic symbiosis with the roots of most land plants. It colonizes root plants and forms tree-shaped subcellular structures. These structures are called arbuscules and exchange nutrients between the fungus and the symbiotic partners. AMF connects the symbionts to a massive hyphal network (can excess 100 m hyphae per cm^3 of soil). This network transports essential nutrients (mainly phosphate), which are difficult for the plant to obtain, and water from AMF to the plant. In exchange for that, AMF receives carbohydrates from the plants. This symbiotic relationship is valuable for both the plant and the fungus and is thought to have played an essential role in global phosphate, and carbon cycling [30].

2.2. State-of-the-Art Image Segmentation

Image segmentation is the process of partitioning an image into different regions (image segments), each representing a distinct object or part of the image. It aims to reduce complexity and/or transform the image into something more meaningful and understandable. Image segmentation can be categorized into semantic, instance, and panoptic segmentation (see figure 2.1). The goal of semantic segmentation is to assign each pixel to the belonging class. Instance segmentation identifies individual objects and classifies those objects according to given classes. Panoptic segmentation combines the concepts of semantic and instance segmentation. It assigns every object a class label and a prediction of the object's identity [25].



(a) image (b) semantic segmentation

(c) instance segmentation (d) panoptic segmentation

Figure 2.1.: Illustration of the three different segmentation methods. Taken from [20].

In the next section, we briefly explain some standard techniques for semantic image segmentation.
Thresholding methods are usually used for gray-scale images. Therefore, it is widely spread in medical applications. Common *thresholding methods* are Otsu, Histogram Concavity Analysis, and Entropy-based methods [35]. Another technique is *K-means clustering*. To initialize the algorithms, k centroids are randomly distributed in the feature space. Then, with each iteration, the centroids are moved until a particular stopping criterion is fulfilled [12]. Additionally, *Support Vector Machines* are frequently used for image segmentation. They try to maximize the margin of the decision hyperplane [3]. A *Markov Random Field* is a probabilistic undirected graphical model. It is a set of random variables that has the Markov property. In the context of image segmentation, it models the dependencies between pixels in the image [16]. These were all more "traditional" methods. Nowadays, mostly *Deep Learning* models are used to achieve state-of-the-art results [23].
The most prominent Deep Learning methods are *Fully Convolutional Neural Networks*. However, neural networks using convolution and deconvolution also achieve excellent results. Neural networks can be pooled with other models, e.g., Conditional Random Fields. This is often used for the last layer of the network. Some other deep learning models used for semantic image segmentation are *Pyramid Methods* [1], *Deep Residual Networks* (ResNet) [13], *Hypercolumns* [11] and *Auto-Encoder* [2]. Research on semantic image segmentation is an ongoing process, and more and more methods are being

researched [23].
Another method for image segmentation, especially semantic segmentation, are Gaussian processes which will be described in the following chapters.

2.3. Decision Trees

A decision tree is a classification and regression model which partitions the input space into N different non-overlapping regions R_N. These regions get a constant or a simple model assigned. A decision tree is generally visualized as a tree (see figure 2.2), where nodes correspond to a feature test, e.g., "worst radius $>$ 16.8". To illustrate this concept, we use the decision tree from figure 2.2. This feature test splits the data into multiple regions. The branches connected to that node are the different outcomes of the corresponding feature test, i.e., in our example, we split the data in "worst radius $>$ 16.8" and "worst radius $\leq$ 16.8". Each leaf is equivalent to the different partitioned regions of the input space. We can see that a decision tree recursively partitions the input space. How to choose an appropriate feature test and construct a decision tree is explained in the next section 2.3.1.

2.3.1. Construction

Constructing an optimal decision tree is NP-complete and thus not feasible with even a moderate number of nodes [18]. Hence, greedy algorithms are used for building a decision tree. The following is adapted from [27].
Generally, we first define a heuristic that chooses the best feature j^* from all class labels $\mathcal{D}$ and the best value t^* from the corresponding class domain τ_i to split. This tuple (j^*, t^*) is defined as follows

$$(j^*, t^*) = \arg \min_{j \in \{1,\dots,D\}} \min_{t \in \tau_j} \text{cost}(\{\mathbf{x}_i, y_i : x_{ij} \leq t\}) + \text{cost}(\{\mathbf{x}_i, y_i : x_{ij} > t\}). \tag{2.1}$$

We evaluate this function recursively on each subtree, starting with the root until a particular stopping heuristic is fulfilled. Such a stopping criterion could be the maximum depth, the cost reduction being too small, or the end distribution being sufficiently small.
In a regression setting, the cost function is defined as

$$\text{cost}(\mathcal{D}) = \sum_{i \in \mathcal{D}} (y_i - \bar{y})^2, \tag{2.2}$$

where $\bar{y}$ is the mean of y.
For a classification setting, there are different heuristics for the cost function. In

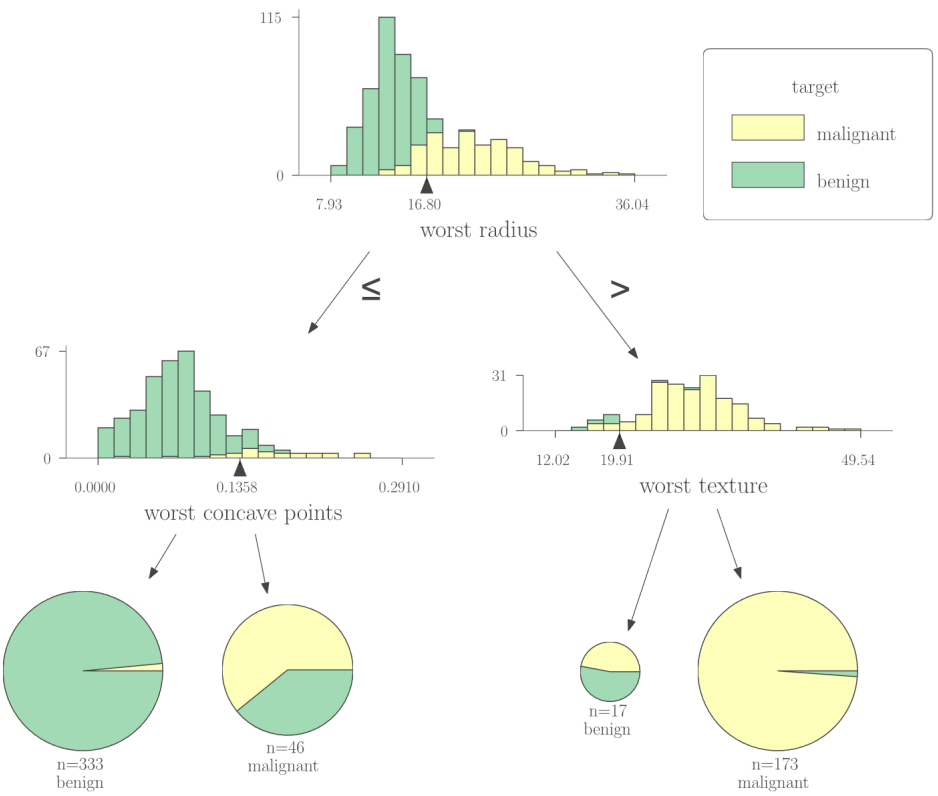


Figure 2.2.: Illustration of a decision tree using the breast cancer data set [5] using [31].

classification problems with $\mathcal{D}$ different class labels, the cost function should measure the impurity of the split. Here, impurity refers to the uncertainty of the resulting data partition. We want to find the split, i.e., a feature test that minimizes the impurity of the resulting two leaves. The distribution of the class labels, $c \in \mathcal{D}$, can be estimated by

$$\hat{\pi}_c = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \mathbb{I}(y_i = c). \tag{2.3}$$

Now we can define according to [27] different impurity measures.

- **Misclassification rate.** We classify the data with the most probable class label $\hat{y}_c = \arg\max_c \hat{\pi}_c$. All other class labels are treated as errors whose rate is defined as

$$\frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \mathbb{I}(y_i \neq \hat{y}) = 1 - \hat{\pi}_{\hat{y}}. \tag{2.4}$$

- **Entropy.** Entropy, in general, measures information. If we want to encode a

probabilistic process, entropy measures the expected number of bits needed to reconstruct the original process. For our purpose, we will define entropy as

$$\mathbb{H}(\hat{\pi}) = -\sum_{c=1}^{C} \hat{\pi}_c \log \hat{\pi}_c. \tag{2.5}$$

Higher entropy of an unknown distribution means that sample values from this distribution are less predictable. Samples from an unknown distribution with lower entropy are more predictable [10].

- **Gini index.**

$$\sum_{c=1}^{C} \hat{\pi}_c (1 - \hat{\pi}_c) \tag{2.6}$$

$\hat{\pi}_c$ is the probability of picking the element and $(1 - \hat{\pi}_c)$ is the probability that the element is misclassified. Consequently, the Gini index measures the expected error rate.

The Gini index and the entropy are more favored as the cost function. We can see in figure 2.3 that the Gini index and the entropy penalize impurity more than the misclassification rate. Empirical studies have shown that it matters only in 2% of the cases whether you choose the Gini index or entropy [34]. One advantage of the Gini index is that it does not rely on the logarithm. Hence, it is cheaper to evaluate, which can lead to a performance boost [27].

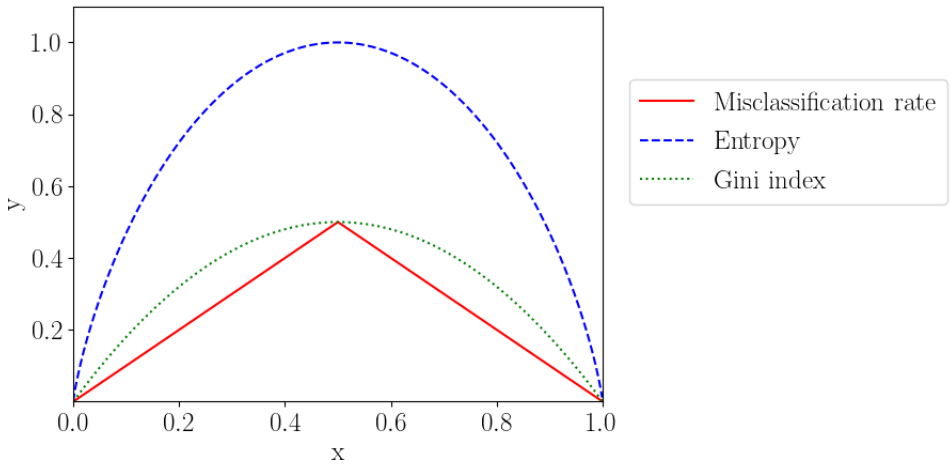


Figure 2.3.: Comparison of the misclassification rate, entropy, and the Gini index.

2.3.2. Random Forests

Without proper pruning or defining a maximum depth, standard decision trees tend to overfit [21]. Therefore, random forests are often used to tackle this issue. Random forests are an ensemble of decision trees. We train M different decision trees each on a bootstrap sample of size N. This is called bagging (bootstrap aggregating). That means we sample $M \times N$ data points from our data with replacement and train M different decision trees on the corresponding sample. For inference, we average over the M decision trees

$$f(x) = \sum_{m=1}^{M} \frac{1}{M} f_m(x). \tag{2.7}$$

This leads to a significant variance reduction. In contrast to decision trees, random forests regrettably lose their interpretability.

2.4. Gaussian Processes

In the book *Gaussian Processes for Machine Learning* [19], Rasmussen defines a Gaussian process as a collection of random variables, any finite number of which have a joint Gaussian distribution. The following sections about Gaussian processes are adapted from Rasmussen's book [19] and [27].
The univariate Gaussian distribution is defined as

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right) \tag{2.8}$$

where μ denotes the mean and σ^2 the variance. The D-dimensional multivariate Gaussian distribution is defined as

$$\mathcal{N}(X|\mu, \Sigma) = \frac{1}{(2\pi)^{D/2}} \frac{1}{\det(\Sigma)^{1/2}} \exp\left(-\frac{1}{2}(X-\mu)^T \Sigma^{-1}(X-\mu)\right) \tag{2.9}$$

where $\mu \in \mathbb{R}^D$ is the mean vector and $\Sigma \in \mathbb{R}^{D \times D}$ is defined as the covariance matrix.

The Gaussian process can be interpreted as a Bayesian prior and posterior. For the prior process, we need to define a mean function

$$m(x) = \mathbb{E}[f(x)] \tag{2.10}$$

and a covariance function

$$k(x, x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))]. \tag{2.11}$$

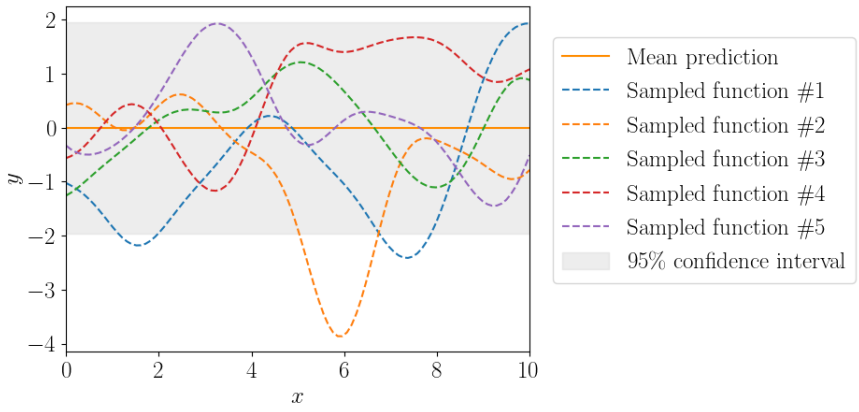


Figure 2.4.: Samples from the prior Gaussian process with a Gaussian kernel.

These two functions completely define the Gaussian process prior and can be expressed as

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')). \tag{2.12}$$

Note that this implies a distribution over functions. We can see this in figure 2.4. This prior process does not depend on data but makes some prior assumptions about the functions. The consistency property is beneficial and defined as

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \right) \implies y_1 \sim \mathcal{N}(\mu_1, \Sigma_{11}). \tag{2.13}$$

Note that this concept also applies to cases where y_1 is a vector of n random variables. Then the applicable submatrix Σ_{11} has shape $n \times n$ and μ_1 is a vector of length n. This will be important later for inferring new data instances from our process.

2.4.1. Gaussian Process Regression

For simplicity reasons, we assume for now that our data is noise free. We can formulate the joint distribution of our training data $\mathbf{f}$ and the test data $\mathbf{f}_*$. According to our prior, the joint distribution is

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X, X) & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right). \tag{2.14}$$

Assuming we have d features, n training points, and n_* test points, then $X \in \mathbb{R}^{n \times d}$ represents the training data, and $X_* \in \mathbb{R}^{n_* \times d}$ is the test data. $K(\cdot, \cdot)$ is the covariance/

kernel function that evaluates the covariance pairwise between the two input matrices, $K(\cdot,\cdot)$, then results again in a matrix. In this case $K(X, X_*) \in \mathbb{R}^{n \times n_*}$. As observed in equation 2.13, we can divide the covariance matrix Σ into four submatrices because of the consistency property. Now we need to restrict all functions generated by the prior to functions that fit our data. We do that by conditioning the joint prior on the observations, which leads to the posterior process

$$\begin{aligned} \mathbf{f}_* | X_*, X, \mathbf{f} &\sim \mathcal{N}\left(\hat{\mu}, \hat{\Sigma}\right), \\ \hat{\mu} &= K(X_*, X)K(X, X)^{-1}\mathbf{f}, \\ \hat{\Sigma} &= K(X_*, X_*) - K(X_*, X)K(X, X)^{-1}K(X, X_*). \end{aligned} \tag{2.15}$$

The covariance matrix $\hat{\Sigma}$ can be interpreted as an uncertainty estimate of the prediction $\hat{\mu}$. In figure 2.5, we can see three sampled functions of the posterior Gaussian process. We can observe that the posterior process is conditioned on the observations, i.e., every observation is contained by every sampled function. In a more realistic setting,

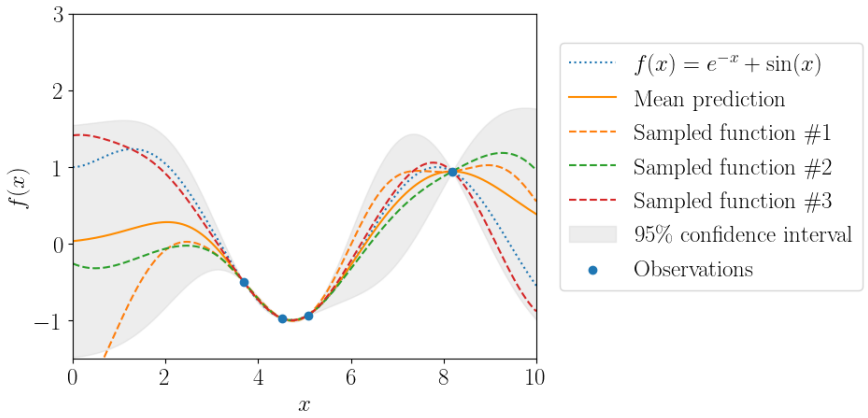


Figure 2.5.: Samples from the posterior Gaussian process with a Gaussian kernel and $f(x) = e^{-x} + \sin(x)$ as the generating process.

data comes with noise. This noise can result from, e.g., measurement errors, human bias, or outliers. Therefore, we now assume noisy observations $\mathbf{y} = f(\mathbf{x}) + \epsilon$ where $\epsilon \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma_n^2)$. Our prior with noise is defined as

$$\mathbb{C}(\mathbf{y}) = K(X, X) + \sigma_n^2 I. \tag{2.16}$$

The joint prior then becomes

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X, X) + \sigma_n^2 I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right), \tag{2.17}$$

and the posterior distribution

$$\mathbf{f}_*|X,\mathbf{y},X_* \sim \mathcal{N}(\bar{\mathbf{f}}_*, \mathbb{C}(\mathbf{f}_*)), \tag{2.18}$$
$$\bar{\mathbf{f}}_* = \mathbb{E}[\mathbf{f}_*|X,\mathbf{y},X_*] = K_*[K+\sigma_n^2 I]^{-1}\mathbf{y},$$
$$\mathbb{C}(\mathbf{f}_*) = K(X_*,X_*) - K_*[K+\sigma_n^2 I]^{-1}K_*.$$

The notation K denotes $K(X,X)$ and K_* is defined as $K(X_*,X)$. With this new conditional distribution, we can compute the posterior Gaussian process with additive Gaussian noise.
As we can observe in equation 2.15, the inversion of $K(X,X)$ is required. Hence, without the use of specialized algorithms, a Gaussian process has a time complexity of $\mathcal{O}(n^3)$ and a memory complexity of $\mathcal{O}(n^2)$ for n data points.

2.4.2. Binary Gaussian Process Classification

As for the Gaussian process regression, we assume a Gaussian process prior over the latent function $f(\cdot)$ with zero mean and the covariance matrix Σ. We are not interested in the particular result of $f(\cdot)$ rather in the result of

$$\pi(\mathbf{x}) = p(\mathbf{y}=1|\mathbf{x}) = \sigma(f(\mathbf{x})). \tag{2.19}$$

Note that because of the sigmoid function, $\pi(\mathbf{x})$ is in the interval $[0,1]$. To infer new data points, we need to define the distribution of the latent function first. Given the training data X, the corresponding binary labels $\mathbf{y} \in \{-1,+1\}$, and the new data points $\mathbf{x}_*$ the distribution is defined as

$$p(f_*|X,\mathbf{y},\mathbf{x}_*) = \int p(f_*|X,\mathbf{x}_*,\mathbf{f})p(\mathbf{f}|X,\mathbf{y})\mathrm{d}\mathbf{f}. \tag{2.20}$$

The posterior distribution of the latent variable $\mathbf{f}$ can be expressed as

$$p(\mathbf{f}|X,\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|X)}{p(\mathbf{y}|X)}. \tag{2.21}$$

To predict a new class label y_* given the training data, their corresponding class labels, and the new data $\mathbf{x}_*$ we need to compute the predictive posterior distribution

$$\pi(\mathbf{x}_*) = p(y_*=+1|X,\mathbf{y},\mathbf{x}_*) = \int \sigma(f_*)p(f_*|X,\mathbf{y},\mathbf{x}_*)df_*. \tag{2.22}$$

Now we do not have a Gaussian likelihood anymore. This leads to analytically intractable integrals, one solution is to approximate the integral. This can be done, e.g., with the **Laplace approximation method** [38] or **expectation propagation** (EP) [26].
To extend this concept to a multiclass classifier, we could use a one-versus-rest or a one-versus-one approach. Another more robust approach for multiclass classification that can also handle noisy input data is proposed by Daniel Hernandez-Lobato [14].

2.4.3. Convolutional Gaussian Processes

In his paper [8], Adrià Garriga-Alonso introduced a convolutional kernel. He has proven that a Gaussian process with a convolutional kernel is equivalent to a convolutional neural network with infinitely many convolutional kernels. This proof can be studied in his paper [8].
Now, we derive the calculations of the convolutional kernel according to [8]. Given an input image matrix $\mathbf{X}$ with height $\mathrm{H}^{(0)}$, width $\mathrm{D}^{(0)}$ and $\mathrm{C}^{(0)}$ channel. The resulting dimensions of $\mathbf{X}$ is then $\mathrm{C}^{(0)} \times \left(\mathrm{H}^{(0)}\mathrm{D}^{(0)}\right)$. Assuming a network with L hidden layers and $i \in \left\{1, \ldots, \mathrm{C}^{(1)}\right\}$ we can define the first linear activations $\mathbf{A}^{(1)}(\mathbf{X})$ as

$$\mathbf{a}_i^{(1)}(\mathbf{X}) = b_i^{(1)}\mathbf{1} + \sum_{j=1}^{\mathrm{C}^{(0)}} \mathbf{W}_{i,j}^{(1)}\mathbf{x}_j, \tag{2.23}$$

where $\mathbf{W}_{i,j}^{(1)}$ is a pseudo-weight matrix of the first activations. To compute the other hidden layers from 2 up to $L+1$, we need to recursively calculate the activations. For layer $\ell+1$ we can calculate the activations by

$$\mathbf{a}_i^{(\ell+1)}(\mathbf{X}) = b_i^{(\ell+1)}\mathbf{1} + \sum_{j=1}^{\mathrm{C}^{(\ell)}} \mathbf{W}_{i,j}^{(\ell+1)}\phi\left(\mathbf{a}_j^{(\ell)}(\mathbf{X})\right). \tag{2.24}$$

The rows of $\mathbf{a}_j^{(\ell+1)}$ consist of the flattened j-th channel of the image after applying the convolutional filter $\phi\left(\mathbf{A}^{(\ell)}(\mathbf{X})\right)$. $\mathbf{A}^{(\ell)}(\mathbf{X})$ has shape $\mathrm{C}^{(\ell)} \times \left(\mathrm{H}^{(\ell)}\mathrm{D}^{(\ell)}\right)$. The output of the last activations $\mathbf{A}^{(L+1)}(\mathbf{X})$ has shape $\mathrm{H}^{(L+1)} = \mathrm{D}^{(L+1)} = 1$, $\mathbf{W}_{i,j}^{(L+1)}$ is then a one-element vector. The pseudo-weight matrix $\mathbf{W}_{i,j}^{(\ell+1)}$ depends on the architecture, but for a convolutional layer, it can be derived from the convolutional filter $\mathbf{U}_{i,j}^{(\ell+1)}$. The μ-th row of $\mathbf{W}_{i,j}^{(\ell+1)}$ corresponds to applying the filter $\mathbf{U}_{i,j}^{(\ell+1)}$ for the μ-th time to $\mathbf{x}_j$. Empty positions of $\mathbf{W}_{i,j}^{(\ell+1)}$ are equal to zero because the filter is not applied there. This concept is also illustrated in figure 2.6.
Using the example from figure 2.6 we can see how this filter $\mathbf{U}_{i,j}^{(0)}$ is applied. We start by applying the filter for the first time, so on the top left corner. That means, we apply A on 1, B on 2, C on 4 and D on 5. That concludes the first column of the first row of the pseudo-weight matrix corresponds to A because we applied the filter on the first element of x_j. The second column of the first row consists of B because B was applied to the second element. The fourth element of x_j was applied with C. Therefore, the fourth column of the first row of the pseudo-weight matrix is C. To apply the filter a

second time, which corresponds to the second row of $\mathbf{W}_{i,j}^{(0)}$, the filter is moved one to the right. Now, the filter is applied to $\left(\begin{smallmatrix}2&3\\5&6\end{smallmatrix}\right)$. So, the second row (second application of filter) of the second column of $\mathbf{W}_{i,j}^{(0)}$ is A. The other entries can be derived equivalently.

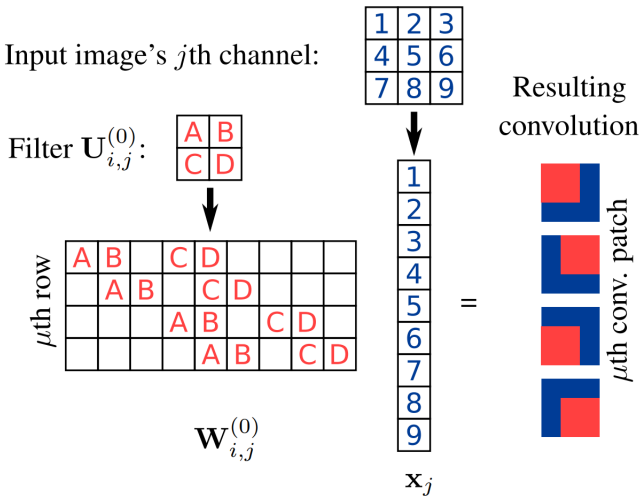


Figure 2.6.: 2D convolution $\mathbf{U}_{i,j}^{(0)}$ of x_j. Taken from [8].

Now we can define the prior over the filters $\mathbf{U}_{i,j}^{(\ell)}$ and biases $b_i^{(\ell)}$ as follows

$$\mathbf{U}_{i,j,x,y}^{(\ell)} \sim \mathcal{N}\left(0, \sigma_w^2 / \mathrm{C}^{(\ell)}\right), \qquad b_i^{(\ell)} \sim \mathcal{N}\left(0, \sigma_b^2\right). \tag{2.25}$$

As mentioned in chapter 2.4, a Gaussian process is completely defined by a mean function and a covariance function. We can first reformulate the activations from 2.24,

$$\mathbf{A}_{i,\mu}^{(\ell+1)}(\mathbf{X}) = b_i^{(\ell+1)} + \sum_{j=1}^{\mathrm{C}^{(\ell)}} \sum_{\nu=1}^{\mathrm{H}^{(\ell)}\mathrm{D}^{(\ell)}} \mathbf{W}_{i,j,\mu,\nu}^{(\ell+1)} \phi\left(\mathbf{A}_{j,\nu}^{(\ell)}(\mathbf{X})\right). \tag{2.26}$$

ℓ and $\ell+1$ denote the input and output layers, j and $i \in \{1, \ldots, \mathrm{C}^{(\ell+1)}\}$ are the input and output channel and ν and $\mu \in \{1, \ldots, \mathrm{H}^{(\ell+1)}\mathrm{D}^{(\ell+1)}\}$ define the indexed position within the feature maps. Now, we can compute the mean by

$$\mathbb{E}\left(\mathbf{A}_{i,\mu}^{(\ell+1)}(\mathbf{X})\right) = \mathbb{E}\left(b_i^{(\ell+1)}\right) + \sum_{j=1}^{\mathrm{C}^{(\ell)}} \sum_{\nu=1}^{\mathrm{H}^{(\ell)}\mathrm{D}^{(\ell)}} \mathbb{E}\left(\mathbf{W}_{i,j,\mu,\nu}^{(\ell+1)} \phi\left(\mathbf{A}_{j,\nu}^{(\ell)}(\mathbf{X})\right)\right) = 0. \tag{2.27}$$

This is derived by using the linearity of the mean and our prior (see 2.25). The prior of $b_i^{(\ell+1)}$ and $\mathbf{W}_{i,j,\mu,\nu}^{(\ell+1)}$ have zero mean and the weights $\mathbf{W}_{i,j,\mu,\nu}^{(\ell+1)}$ are independent of the

previous layer. Hence the mean is zero.
The underlying covariance function

$$\mathbb{C}\left(\mathbf{A}_{i,\mu}^{(\ell+1)}(\mathbf{X}), \mathbf{A}_{i,\mu}^{(\ell+1)}(\mathbf{X}')\right) = \mathbb{V}\left(\mathrm{b}_i^{(\ell)}\right) + \sum_{j=1}^{\mathrm{C}^{(\ell)}} \sum_{j'=1}^{\mathrm{C}^{(\ell)}} \sum_{\nu=1}^{\mathrm{H}^{(\ell)}\mathrm{D}^{(\ell)}} \sum_{\nu'=1}^{\mathrm{H}^{(\ell)}\mathrm{D}^{(\ell)}} \mathbb{C}\left(\mathbf{W}_{i,j,\mu,\nu}^{(\ell+1)} \phi\left(\mathbf{A}_{j,\nu}^{(\ell)}(\mathbf{X})\right), \mathbf{W}_{i,j',\mu,\nu'}^{(\ell+1)} \phi\left(\mathbf{A}_{j',\nu'}^{(\ell)}(\mathbf{X}')\right)\right) \tag{2.28}$$

can be simplified by using the prior assumptions, namely, zero mean, the prior distributions, and independence to

$$\mathbb{C}\left(\mathbf{A}_{i,\mu}^{(\ell+1)}(\mathbf{X}), \mathbf{A}_{i,\mu}^{(\ell+1)}(\mathbf{X}')\right) = \sigma_b^2 + \sum_{j=1}^{\mathrm{C}^{(\ell)}} \sum_{j'=1}^{\mathrm{C}^{(\ell)}} \sum_{\nu=1}^{\mathrm{H}^{(\ell)}\mathrm{D}^{(\ell)}} \sum_{\nu'=1}^{\mathrm{H}^{(\ell)}\mathrm{D}^{(\ell)}} \mathbb{E}\left(\mathbf{W}_{i,j,\mu,\nu}^{(\ell+1)} \mathbf{W}_{i,j',\mu,\nu'}^{(\ell+1)}\right) \mathbb{E}\left(\phi\left(\mathbf{A}_{j,\nu}^{(\ell)}(\mathbf{X})\right) \phi\left(\mathbf{A}_{j',\nu'}^{(\ell)}(\mathbf{X}')\right)\right). \tag{2.29}$$

For different channels two weights $\mathbf{W}_{i,j}^{(\ell+1)}$ and $\mathbf{W}_{i,j'}^{(\ell+1)}$ where $j \neq j'$ are independent, so $\mathbb{E}\left(\mathbf{W}_{i,j,\mu,\nu}^{(\ell+1)} \mathbf{W}_{i,j',\mu,\nu'}^{(\ell+1)}\right) = 0$ for $j \neq j'$. As we can see in figure 2.6 $\mathbf{W}_{i,j}^{(\ell+1)}$ only contains independent variables or zeros. This concludes that $\mathbb{E}\left(\mathbf{W}_{i,j,\mu,\nu}^{(\ell+1)} \mathbf{W}_{i,j',\mu,\nu'}^{(\ell+1)}\right) = 0$ for $\nu \neq \nu'$. With this, we can re-express the equation by removing the sums over j' and μ':

$$\mathbb{C}\left(\mathbf{A}_{i,\mu}^{(\ell+1)}(\mathbf{X}), \mathbf{A}_{i,\mu}^{(\ell+1)}(\mathbf{X}')\right) = \sigma_b^2 + \sum_{j=1}^{\mathrm{C}^{(\ell)}} \sum_{\nu=1}^{\mathrm{H}^{(\ell)}\mathrm{D}^{(\ell)}} \mathbb{E}\left(\mathbf{W}_{i,j,\mu,\nu}^{(\ell+1)} \mathbf{W}_{i,j,\mu,\nu}^{(\ell+1)}\right) \mathbb{E}\left(\phi\left(\mathbf{A}_{j,\nu}^{(\ell)}(\mathbf{X})\right) \phi\left(\mathbf{A}_{j,\nu}^{(\ell)}(\mathbf{X}')\right)\right). \tag{2.30}$$

To simplify it even further, we can use the fact that the μ-th row of $\mathbf{W}_{i,j,\mu,\nu}^{(\ell+1)}$ must be zero for indices ν which do not belong to the μ-th convolutional patch. Hence, the sum over μ can be restricted to a non-zero interval. The covariance of the first layer is

$$K_\mu^{(1)}(\mathbf{X}, \mathbf{X}') = \mathbb{C}\left(\mathbf{A}_{i,\mu}^{(1)}(\mathbf{X}), \mathbf{A}_{i,\mu}^{(1)}(\mathbf{X}')\right) = \sigma_b^2 + \frac{\sigma_w^2}{\mathrm{C}^{(0)}} \sum_{i=1}^{\mathrm{C}^{(0)}} \sum_{\nu \in \mu\text{-th patch}} X_{i,\nu} X'_{i,\mu}. \tag{2.31}$$

For the other layers, we have

$$K_\mu^{(\ell+1)}(\mathbf{X}, \mathbf{X}') = \mathbb{C}\left(\mathbf{A}_{i,\mu}^{(\ell+1)}(\mathbf{X}), \mathbf{A}_{i,\mu}^{(\ell+1)}(\mathbf{X}')\right) = \sigma_b^2 + \sigma_w^2 \sum_{\nu \in \mu\text{-th patch}} V_\nu^{(\ell)}(\mathbf{X}, \mathbf{X}'), \tag{2.32}$$

where

$$V_\nu^{(\ell)}(\mathbf{X}, \mathbf{X}') = \mathbb{E}\left(\phi\left(\mathbf{A}_{j,\nu}^{(\ell)}(\mathbf{X})\right) \phi\left(\mathbf{A}_{j,\nu}^{(\ell)}(\mathbf{X}')\right)\right). \tag{2.33}$$

This is the covariance of the activations. With this covariance, we can compute the actual kernel in a closed form. First, we choose an activation function. Here, we pick the ReLU function ($\phi(x) = \max(0, x)$). We can obtain a closed form for this by

$$V_\nu^{(\ell)}(\mathbf{X}, \mathbf{X}') = \frac{\sqrt{K_\nu^{(\ell)}(\mathbf{X}, \mathbf{X}) K_\nu^{(\ell)}(\mathbf{X}', \mathbf{X}')}}{\pi} \left(\sin \theta_\nu^{(\ell)} + \left(\pi - \theta_\nu^{(\ell)} \right) \cos \theta_\nu^{(\ell)} \right), \tag{2.34}$$

where $\theta_\nu^{(\ell)} = \cos^{-1}\left(K_\nu^{(\ell)}(\mathbf{X}, \mathbf{X}') / \sqrt{K_\nu^{(\ell)}(\mathbf{X}, \mathbf{X}) K_\nu^{(\ell)}(\mathbf{X}', \mathbf{X}')} \right)$.
With the mean function from equation 2.27 and the covariance function from equations 2.31/2.32, we completely defined a Gaussian process with a convolutional kernel.

2.4.4. Treed Gaussian Processes

The main disadvantage of Gaussian processes is the cubic complexity. Hence, for large data sets, Gaussian processes become intractable on current hardware. That is why there is much research in this area. Nowadays, there are many different approaches to tackle this problem. A widespread practice are conditional independence assumptions [33] on a predefined set of variables. Another method is partitioning the input space. Then, only the kernel matrices on the corresponding subset of the whole data set must be calculated. In his dissertation [9], Gramacy proposes using treed Gaussian processes to tackle the complexity issue. A treed Gaussian process is a scalable alternative and takes advantage of partitioning the input space. It uses a decision tree to split the input space. Then, for each data partition (leaf node of the decision tree), we train one Gaussian process on this specific data partition.
To construct a treed Gaussian process, we first start in the root node of the decision tree containing the whole training data. Then we recursively build the decision tree until a particular stopping criterion is fulfilled. In the end, each leaf contains one data partition/cluster. After that, we train a Gaussian process for each leaf on the corresponding data partition (see figure 2.7).
To predict a new data instance, we first traverse the decision tree to the corresponding leaf node to find its Gaussian process. Then, we use this specific Gaussian process of this particular leaf to compute the prediction. This whole process is illustrated in figure 2.7.
A treed Gaussian process has a tremendously improved complexity compared to the normal Gaussian process. We define ℓ as the maximum number of data points per leaf node across all leaves. Then, the complexity of training a treed Gaussian process is $\mathcal{O}(n \log n + n\ell^2)$, and the complexity of predicting a new data instance is $\mathcal{O}(\log n + \ell)$. To take advantage of this improved complexity, the decision tree must be homogeneous, i.e., the number of data points in each leaf should be roughly the same [7].

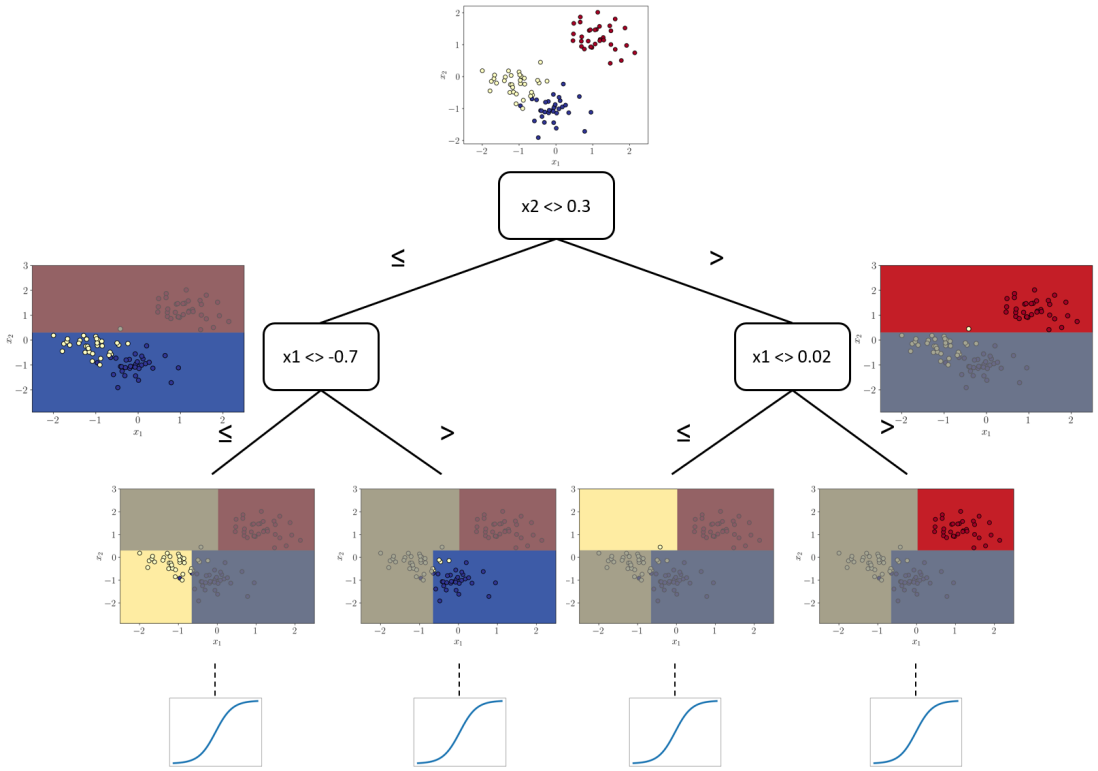


Figure 2.7.: The decision tree recursively partitions the feature space. The data points outside the gray-shaded area are the current data points which will be further divided. In the end, there are four data partitions. For each of these partitions, one Gaussian process is trained.

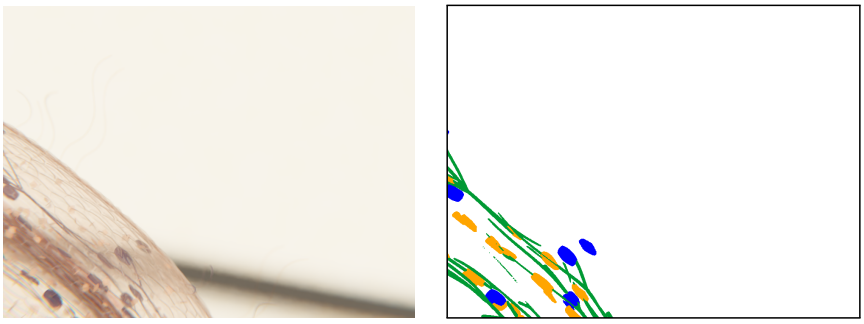
Fröhlich et al. [6, 7] proposed using random forests instead of decision trees to avoid overfitting. Here, multiple trees are trained simultaneously on a bootstrap sample (see chapter 2.3). To predict a new data instance for a random forest Gaussian process, we sum up the result of all trees. Then we follow the standard one-vs-rest classifier setting and predict the class with the maximum value. That leads to a training complexity of $\mathcal{O}(Tn \log n + Tn\ell^3)$ and a prediction complexity of $\mathcal{O}(T \log n + T\ell)$ for T trees [6].
In conclusion, treed Gaussian processes offer a robust and scalable alternative to the traditional Gaussian process.

3. Image Segmentation using treed Gaussian Processes

In this chapter, we explain the algorithm of treed Gaussian processes in the context of image segmentation and its implementation. Then, we evaluate this approach on a customized and extended MNIST data set and on the light microscopy image data set.

3.1. Light Microscopy Data Set

We use artificially rendered light microscopy images of Arbuscular Mycorrhiza Fungi (see chapter 2.1) from Jan Watter's master's thesis [37]. We do not use real light microscopy images because it is tough to obtain a high number of images to train a machine learning model sufficiently. Through a Blender file, as many images as desired can be rendered. Every image is of size 736×973 pixels. In figure 3.1, we can see an example from this data set. We have four different classes: background (white), hyphae (green), arbuscules (orange), and vesicles (blue).



a) training image b) fully segmented image

Figure 3.1.: Example from the light microscopy image data set from [37].

3.2. Treed Gaussian Processes for Image Segmentation

Now, we apply the concept of treed Gaussian processes from chapter 2.4.4 to semantic image segmentation. The data consists of N images of shape (H, W, C), where H is the height, W is the width, and C is the number of channels of the image. We treat every pixel in every channel of the image as one feature. Hence, we have $H \cdot D \cdot C$ features per image. Furthermore, we could also use further feature extraction for images, e.g., edge detection or different filters. Now, these feature vectors are used to train a decision tree. The decision tree uses the individual pixel of a specific channel of the images for the feature tests. As explained in chapter 2.4.4, the decision tree partitions the input space. In the context of images, the decision tree clusters similar images in the leaf nodes (see figure 2.7). One Gaussian process is trained for every leaf node (cluster). We use the convolutional kernel presented in chapter 2.4.3 to calculate the kernel matrices. Using a convolutional kernel allows us to capture the neighboring pixels' dependencies. To segment a new image instance, the image traverses the decision tree to determine the

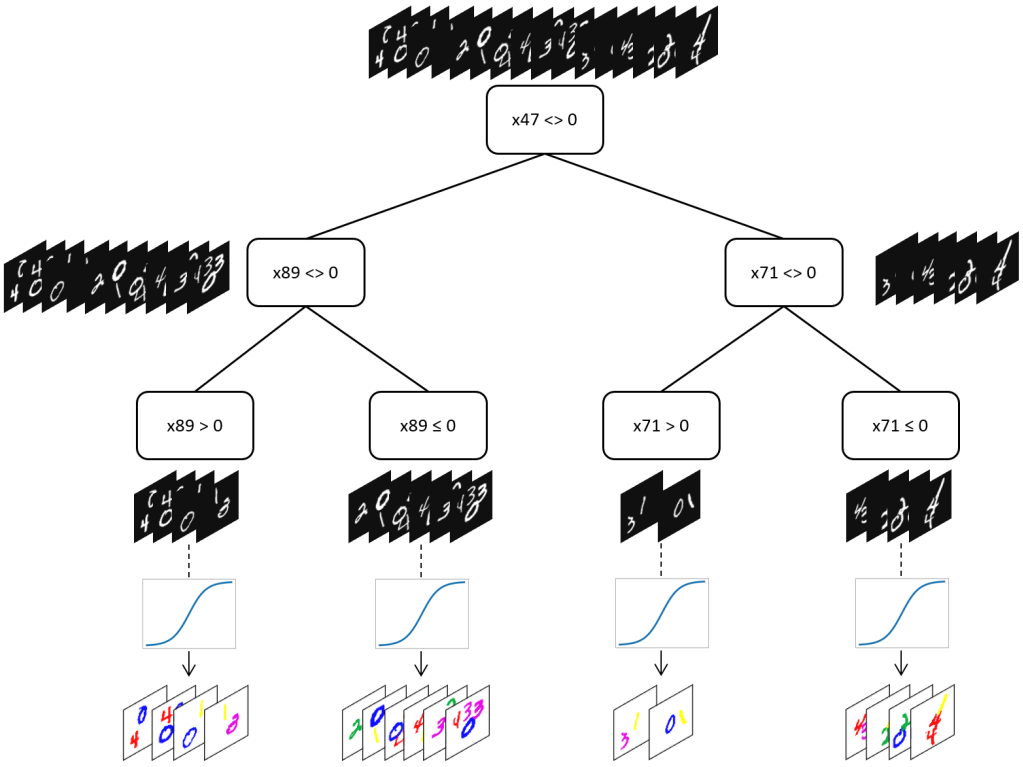


Figure 3.2.: Treed Gaussian process with exemplary data, feature tests, and clusters.

corresponding leaf node. We calculate the kernel matrix $K(X_*, X)$ for the corresponding Gaussian process. Then we calculate the segmented image by $K(X_*, X)K(X, X)^{-1}\mathbf{f}$ (see equation 2.17).

3.3. Performance metrics

By evaluating the performance of a classification model, we compare the model's output with the groundtruth. In the context of image segmentation, the comparison is usually made pixel-wise. In this section, we present several performance metrics, which we will later use to determine our model's performance.

		Predicted Class	
		positive	negative
Actual Class	positive	TP (True positive)	FN (False negative)
	negative	FP (False positive)	TN (True negative)

Figure 3.3.: Confusion matrix for binary classification [17]

.

Precision defines the proportion of true positive predictions out of all positive predictions made by the model

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{3.1}$$

High precision means that the model is not classifying many false positives or is quite confident when making positive predictions
Recall is the fraction of true positives out of all actual positive instances

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{3.2}$$

A high recall indicates that the model properly identifies the majority of the positive observations correctly
Both precision and recall are usually part of measuring the performance of a classification model. The trade-off between them is often essential, as a highly precise model could miss a lot of relevant results (low recall). In contrast, a model with high recall might return many false results (low precision).
The **F1-Score** tries to measure a good balance between precision and recall. It is defined

as the harmonic mean of precision and recall

$$\mathrm{F1} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}. \tag{3.3}$$

The **Jaccard Index** [22, 29] (also known as Intersection over Union) defines, in general, the similarity of two sets A and B. In the context of image segmentation, the Jaccard Index can be interpreted as the area of overlapping regions. Let A be the groundtruth and B the model's output image. Then the Jaccard Index is defined as

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}. \tag{3.4}$$

Equations 3.1, 3.2, 3.3 are taken from [17].

3.4. Implementation

In this section, we describe how the treed Gaussian process for image segmentation is implemented. First, we start with the data-preprocessing. Then we explain how the training and the image segmentation (prediction) are implemented.

3.4.1. Data-Prepocessing

We start with a training set. The training data has shape (N, H, W, C), where N is the number of samples, H and W describe the height and width of the image, and C is the number of channels. First, we use a one-hot encoding to handle multiple classes. That means the groundtruth of the training data has shape (N, H, W, C, y) where y is the number of different class labels. One-hot encoding is a preprocessing technique to convert categorical values to numerical values. This is done by representing the categorical values by a binary vector, where each category represents one entry in the binary vector. The position of the "1" indicates the class. For example, we have a categorical variable *destination* $\in \{\text{Berlin}, \text{Munich}, \text{Hamburg}\}$. Then, the value "Berlin" is represented by $(1, 0, 0)$, "Munich" by $(0, 1, 0)$, and "Hamburg" by $(0, 0, 1)$.
The complete images are not used directly. The images are being upsampled. We use a window of size (H_p, W_p) pixels and move this window by s (stride) pixels in x and y direction. That results in $N_p \gg N$ overlapping images of size (H_p, W_p) (see figure 3.4). We call these smaller images patches. This brings the advantage that the Gaussian process only sees little context. This leads to better generalization, consequently, to better results. Many image patches are mostly background. Therefore, we subsample the patches such that images with mainly background are ignored except for one.

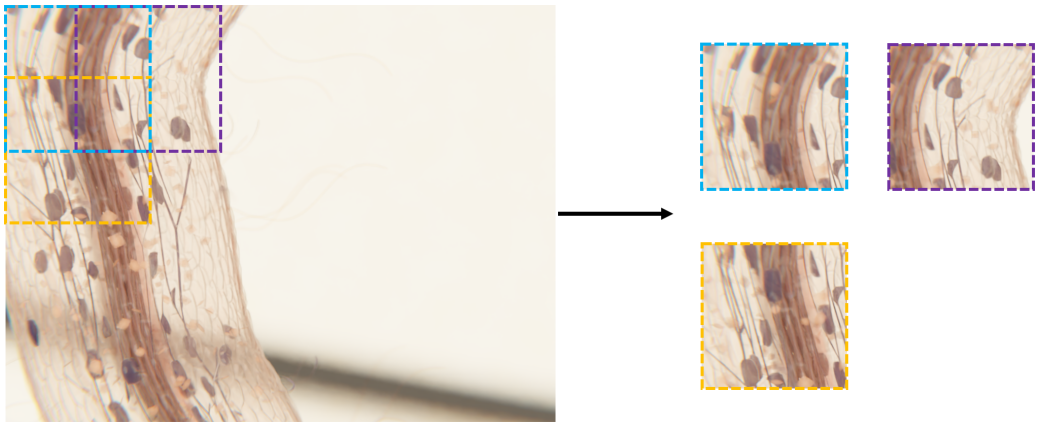


Figure 3.4.: Illustration of the patching process.

3.4.2. Model Training

First, a decision tree is trained using the image patches. We use an optimized CART decision tree from scikit-learn [32]. As the cost function, we apply the Gini index.
For each data partition (leaf), we train one Gaussian process on this particular data partition. The implementation from Adrià Garriga-Alonso [8] is used to calculate the convolutional kernel function. This library transforms a neural network architecture into a kernel function. This kernel calculates the covariance matrices $K(X, X)$ and $K(X_*, X)$. We can speed up the calculations of the kernel matrices by using the symmetric property of kernels. From the definition of a kernel

$$k(x, x') = \langle \phi(x), \phi(x') \rangle, \tag{3.5}$$

we can see that a kernel is a symmetric function [3]. We can exploit this by only calculating the upper triangular matrix. Then we can mirror the upper matrix to get the lower triangular matrix, i.e., $K(X, X)_{(1,2)} = K(X, X)_{(2,1)}$. This roughly halves the computation time of the kernel matrix. The kernel matrix is calculated in batches step by step and not directly on the whole data. This is done to deal with the high dimensionality of the data. The kernel matrix can be computed on the GPU to speed up the computation time further.
Both the decision tree and the kernel matrices $K(X, X)$ are stored in files and can be loaded such that the tree structure and the kernel matrices do not need to be calculated again.
Several different hyperparameters need to be optimized. In table 3.1, we summarize all important hyperparameters. All these hyperparameters can be determined for each data set using, e.g., cross-validation.

Table 3.1.: Hyperparameters of a treed Gaussian process with a convolutional kernel.

Neural Network Architecture	**Decision Tree**	**Preprocessing**
number of layers	maximum depth	patch size
activation functions	cost function	stride
kernel size for each layer		
variance bias		
variance weight		

3.4.3. Inference

To apply the segmentation on a new image, the whole input image is divided into patches, as described in chapter 3.4.1. Segmenting a new image is done by first traversing the decision tree to find the predicted leaf node, thus, the corresponding Gaussian process. Using this Gaussian process, we segment the image patch. As we can observe in equation 2.17, we must invert the kernel matrix $K(X, X)$, but we do not want to explicitly invert the kernel matrix. Instead, we rewrite the mean of the posterior process as

$$\begin{aligned} \hat{\mu} &= K(X_*, X)c, \\ \text{where } c &= K(X, X)^{-1}\mathbf{f}. \end{aligned} \tag{3.6}$$

Now, we solve the linear system $K(X, X)c = \mathbf{f}$ to obtain c. We use a trick to improve

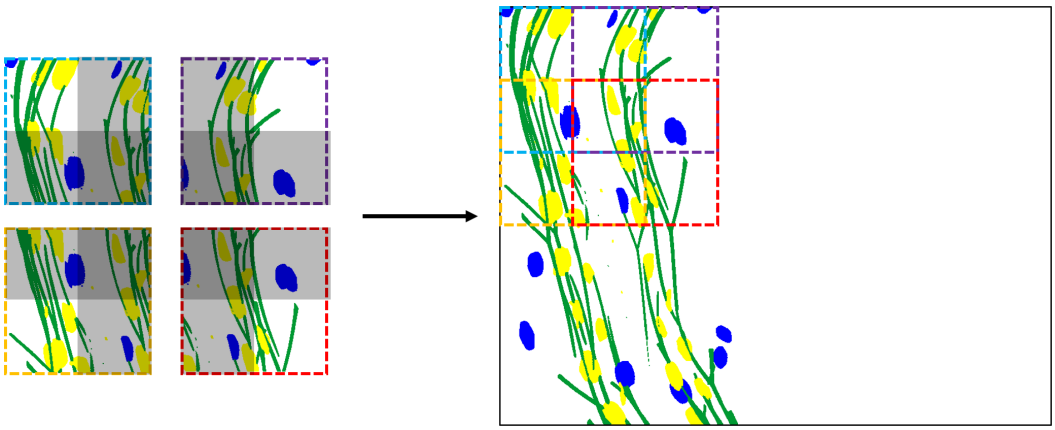


Figure 3.5.: Reconstruction of the whole image using patches. The light gray shaded regions are added together by two patches. Whereas the dark gray areas are added up by all four patches.

the computation time and numerical accuracy [15]. We apply least squares to solve the system of linear equations. The calculations are done by the library SciPy [36] and CuPy [28], depending on if it is calculated on the CPU or the GPU. Then we compute the matrix multiplication $\mathbf{f}_* = K(X_*, X)c$. That result is then squashed through a sigmoid function. This process is done with every patch of the image. After that, we combine all patches by adding up the individual one-hot vectors of the corresponding pixel. Note that the offset for each image patch will be considered in this sum (see figure 3.5). Finally, we take the arg max over every pixel's one-hot vector to predict the pixel's class label. This whole process is visualized in figure 3.6.

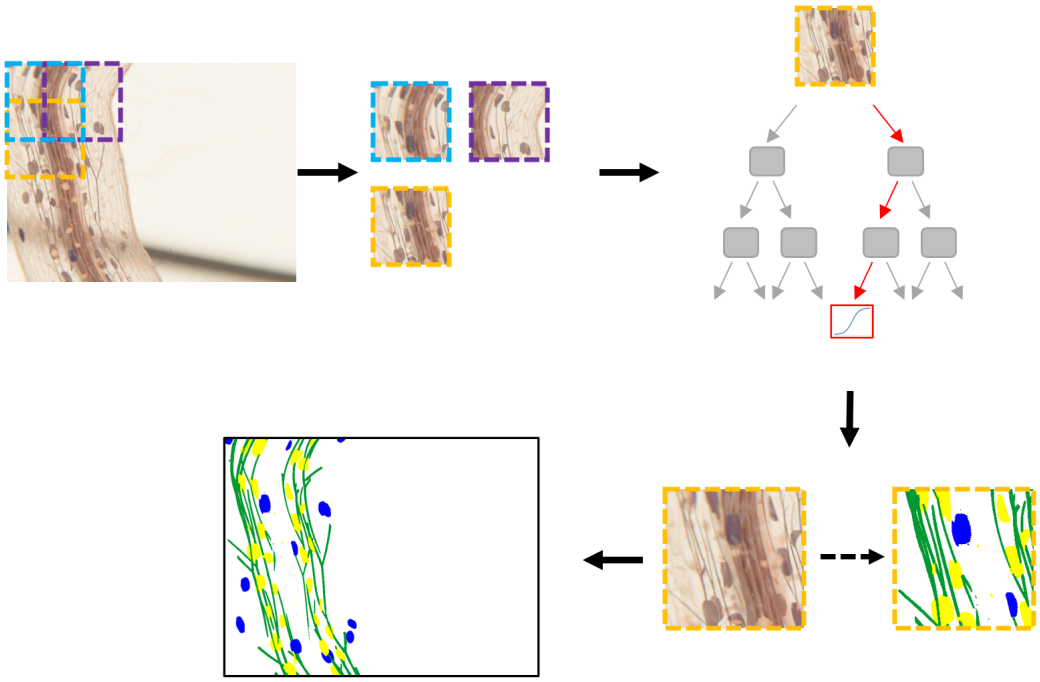


Figure 3.6.: Flow diagram of the segmentation of a new image. First, we divide the image into patches. Then the image traverses the decision tree to find the corresponding Gaussian process. Using the calculations from equation 3.6 we obtain the segmented patch. Then all patches are added up, and the arg max is taken to get the final segmentation.

3.4.4. GPU optimizations

The most expensive operations can be calculated on the GPU. We use the library CuPy [28] to calculate the operations on the GPU. The most expensive operations are:

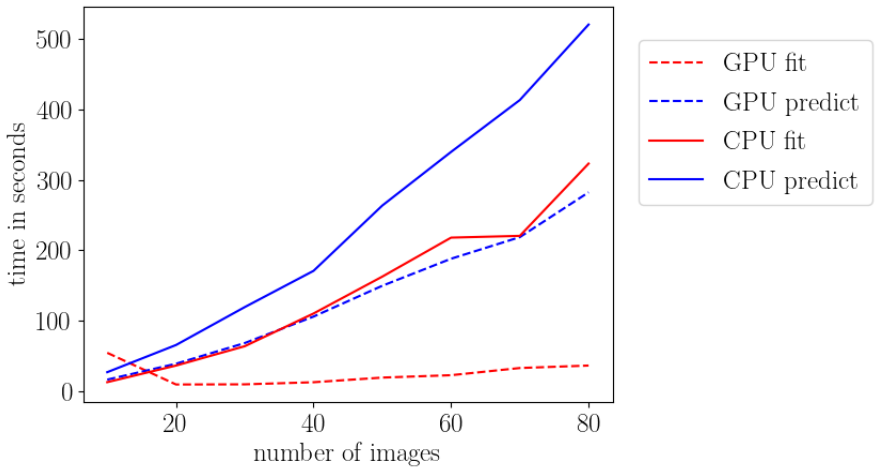


Figure 3.7.: Comparison of calculating the treed Gaussian process on the CPU and the GPU. Used 80/20 training-test split, with the extended MNIST data set and the hyperparameters from table 3.2. The time to train (fit) the model and the time to predict the test data set are benchmarked.
Hardware: Ryzen 7 1700 @ 3.00 GHz, 16GB Ram, GTX 1080, WSL2.

calculating the kernel matrix, solving the least-squares problem from equation 3.6, and the matrix multiplication. We trained multiple treed Gaussian processes on the extended MNIST data set (see section 3.5) with an 80/20 training-test split, i.e., 20% of the images are in the test set. Moreover, we used the same hyperparameters from table 3.2. We benchmarked the time to fit the model and predict the test images using the first time the GPU and the second time using only the CPU. In figure 3.7, we can see that the computation time is tremendously faster using the GPU. The more images are used, the more significant the difference between the CPU and the GPU. The reason for the smaller difference when using fewer images is the overhead generated by the GPU. The image patches must be transferred from the CPU to the GPU. If only a few images are transferred, the actual computation time on the GPU is negligible compared to the transfer time. So, the more images used, the smaller the relative difference between the transfer and the computation time. For ten images, the CPU is much faster. This could be due to some uncontrollable external factors.

3.5. Evaluation on the extended MNIST Data Set

First, we evaluate the performance of our model using simpler images. We use the extended MNIST data set from [24]. In figure 3.8, we can see that this data set combines

different images of the original MNIST data set in a new image. The original MNIST data set [4] consists of images with single handwritten digits ranging from 0 to 9. All images have shape 60×60 pixels. In our case, we only use digits from 0 to 4 to reduce complexity. Overall, we have 6 classes, 5 for the digits and one for the background. The classes are one-hot encoded. We used the standard implementation as described in

a) training images b) fully segmented images

Figure 3.8.: Example images from the extended MNIST data set from [24].

Table 3.2.: Hyperparameters for the extended MNIST experiment.

Preprocessing	
patch size	20×20
stride	5

Decision Tree	
maximum depth	3
cost function	Gini index

Neural Network	
number of layers	4
activation functions	ReLU
kernel sizes	[15,3,3,20]
variance bias	7.86
variance weight	136.71

section 3.4. The treed Gaussian process is trained using 200 images. The performance is evaluated using 100 images. The hyperparameters used for the model can be found in table 3.2. As we can see in table 3.3, our model achieved great results on this data set. In figure 3.9, we can visually compare the results of the model. In appendix 1, we can find further comparisons.
We can see in figure 3.9 that our model has excellent shape recognition and very good class predictions. Some parts of the digits get misclassified. This is due to the intrinsic model's inaccuracy. Another reason for this could be the image patches. The stride and the patch size are chosen so the Gaussian process can see the whole digits at least once. However, the Gaussian process only sees parts of the whole image. The model sees a

Table 3.3.: Evaluation of the treed Gaussian process. It was trained using 200 images and evaluated on 100 images. The model used the hyperparameters from table 3.2. All numbers are rounded to 2 decimals.

	Background	Class 0	Class 1	Class 2	Class 3	Class 4
Precision	0.98	0.80	0.87	0.84	0.85	0.90
Recall	1.00	0.82	0.81	0.61	0.74	0.62
F1-Score	0.99	0.81	0.84	0.70	0.80	0.74
Jaccard Index	0.98	0.68	0.72	0.55	0.66	0.58
Mean F1-Score	0.81					

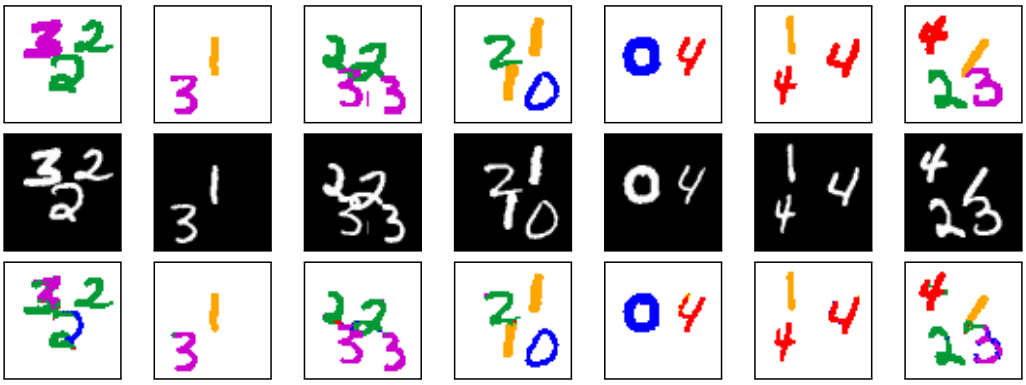


Figure 3.9.: Comparison of the output of the treed Gaussian process trained on the extended MNIST data set. Top: groundtruth, middle: original, bottom: model output.

part of the digit and is more confident about its prediction than seeing the whole digit. For example, we can take a look at the leftmost images. The arc of the lower two gets misclassified. It looks like a zero cut in half vertically. Now, there is likely an image patch that only contains the arc of the zero. Consequently, the model predicts this as a zero with high confidence.

3.6. Evaluation of the light microscopy images

Now, we evaluate the performance of the treed Gaussian process on the light microscopy data set from section 3.1. We have several new challenges with this data set. Firstly, the images are more complex than the extended MNIST data set. The images are much finer and more sparse than the previous data set. Secondly, in contrast to the

extended MNIST data set, we have colored images now. The most significant change is the image resolution. The images are of size 736×973 pixels. That results in 716.128 features, nearly $200\times$ more features than the MNIST data set (3600 features). Assuming we use patches of size 256×256 pixel, we need to construct a decision tree with over 65.000 features. Constructing a decision tree with these many features is very expensive/infeasible. One way to solve this problem is using smaller patch sizes, which is not really applicable in this case because the Gaussian process must see more context for good predictions. Another reason is if we are using image patches with smaller sizes, the number of image patches increases. Consequently, the kernel matrices become bigger, which leads to more computational costs. So, we need to tackle this problem in another way.
We propose to use principal component analysis (PCA) [3] to reduce the dimensionality of the data when constructing the decision tree. That means that we project the image patches into a lower dimensional space using PCA. Then, we use this lower dimensional data to build the decision tree. After that, we use the original (high dimensional) data to train the corresponding Gaussian processes in each leaf. To segment a new image, we first use the already trained PCA to project the new image patches into the lower dimensional space. This lower dimensional patch traverses the decision tree to find the corresponding Gaussian process. Then, we use the original image patch and calculate the segmentation using the specific Gaussian process. We follow this for all obtained image patches.
Using three principal components, we retain 99.9% of the original variance. In appendix 2, we can see that using the low dimensional data obtained by PCA to train the decision tree instead of the original data does not make a significant difference.
We used 20 images (2340 image patches of size 256×256 pixels) to train our model with the parameters from table 3.4.

Table 3.4.: Hyperparameters for the light microscopy images experiment.

Preprocessing	
patch size	256×256 pixel
stride	64

Decision Tree	
maximum depth	3
cost function	Gini index

Neural Network	
number of layers	4
activation functions	ReLU
kernel sizes	[30,3,3,256]
variance bias	7.86
variance weight	136.71

We obtained acceptable results. In table 3.5, we can see that our model achieved excellent results for the background and acceptable results for the other three classes. Furthermore, in figure 3.10, we can visually compare the original images with the groundtruth and the segmentation of our model. In appendix 3, we can find further segmentation examples of our model.

Table 3.5.: Evaluation of the treed Gaussian process. It was trained using ten images and evaluated on four images. The model used the hyperparameters from table 3.4. All numbers are rounded to 2 decimals.

	Background	Hyphae	Arbuscules	Vesicles
Precision	0.96	0.53	0.69	0.75
Recall	0.99	0.30	0.42	0.23
F1-Score	0.97	0.39	0.52	0.36
Jaccard Index	0.95	0.23	0.35	0.22
Mean F1-Score	0.56			

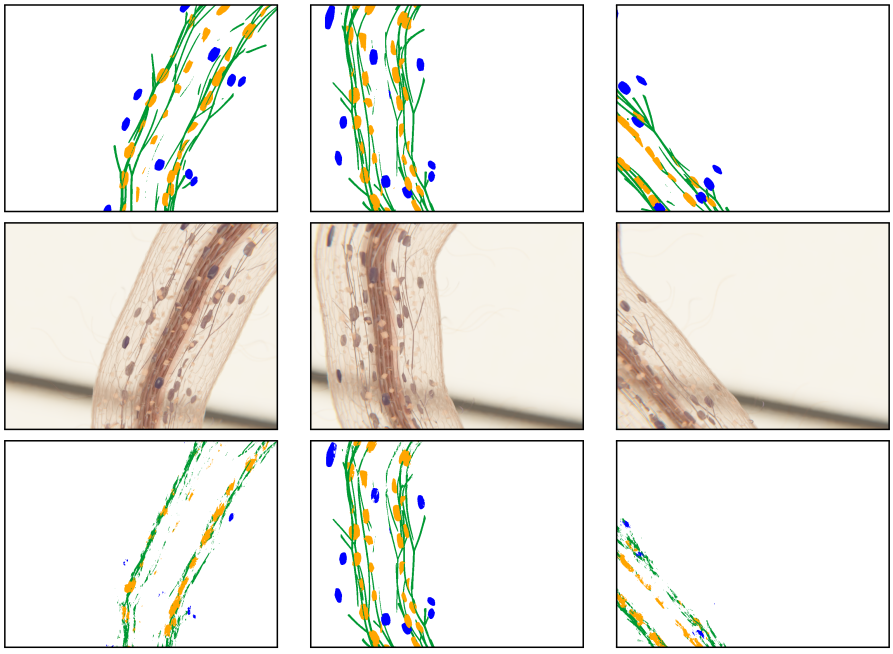


Figure 3.10.: Comparison of the output of the treed Gaussian process trained on the light microscopy image data set. Top: groundtruth, middle: original, bottom: model output.

We can see in figure 3.10 that our model recognizes the general structure of the image well. Unfortunately, the predictions are scattered and very noisy. Furthermore, we can observe that the recall of especially vesicles (blue) is poor. One reason for the prediction quality might be the image's contrast. The contrast of the images is not very high regarding the different class labels, especially for the hyphae.
We can observe an outstanding result in the middle column of figure 3.10. In contrast, to the other predictions, it is not scattered, and almost every pixel is segmented correctly. This might be because the image patches almost exactly matched the training images. As mentioned earlier, the images have been rendered using Blender. Hence, there can be a very high similarity between the images.
The main reason for this big performance difference between the extended MNIST data set and this one is the underlying image structure. The microscopy images are far more complex. The distinct classes are much finer and sparse.

4. Conclusion

This thesis contains a description of treed Gaussian processes for light microscopy image segmentation. In the beginning, we introduced the concept of treed Gaussian processes. We explained the foundations of decision trees and Gaussian processes. We analyzed the connection between Gaussian processes and convolutional neural networks. We have seen that treed Gaussian processes offer a scalable alternative to the traditional Gaussian processes. Next, we evaluated our model on the extended MNIST data set. We achieved great results on that data set. We benchmarked our model's performance on the light microscopy images. At this point, we encountered various challenges. We obtained acceptable results on the light microscopy images. Unfortunately, the segmentations were scattered. The main reason for the segmentation quality was the intrinsic complex structure of the images, even though the model approximated the general pattern well.
The treed Gaussian process analyzed in this thesis can be used to segment microscopy images of medium size with 16-32GB RAM (depending on the tree structure and patch sizes), a Ryzen 7 1700, and a GTX 1080. Semantic image segmentation can be done very efficiently using our model with a training data size of 20 images. We can apply our model to other images as well as it is not limited to light microscopy images.
For future work, we developed some ideas to improve the model. We used the Gini index as the cost function for the decision tree. The Gini index only uses one feature. Instead of the Gini index, we could use a convolutional cost function, i.e., a heuristic that uses not only one pixel but also considers the neighboring pixel. That may lead to better clustering. Another idea is to use an utterly different clustering/data partitioning method instead of the decision tree, e.g., t-SNE. Furthermore, we have seen that the Gaussian process provides an uncertainty estimate for predictions. We can extend our model to incorporate these uncertainty estimates, potentially leading to better segmentations. We obtained scattered predictions for the light microscopy images. Therefore, we propose using a convolutional layer as the last step after the Gaussian process to solve this issue. That should produce smoother and less noisy predictions.
Overall, we have seen that a treed Gaussian is an accurate and scalable model for image segmentation tasks.

Bibliography

[1] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden. "Pyramid methods in image processing." In: *RCA engineer* 29.6 (1984), pp. 33–41.

[2] Y. Bengio et al. "Learning deep architectures for AI." In: *Foundations and trends® in Machine Learning* 2.1 (2009), pp. 1–127.

[3] C. M. Bishop and N. M. Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[4] L. Deng. "The mnist database of handwritten digit images for machine learning research." In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[5] D. Dua and C. Graff. *UCI Machine Learning Repository*. 2017.

[6] B. Fröhlich, E. Rodner, M. Kemmler, and J. Denzler. "Efficient Gaussian process classification using random decision forests." In: *Pattern Recognition and Image Analysis* 21.2 (2011), pp. 184–187.

[7] B. Fröhlich, E. Rodner, M. Kemmler, and J. Denzler. "Large-scale gaussian process multi-class classification for semantic segmentation and facade recognition." In: *Machine vision and applications* 24.5 (2013), pp. 1043–1053.

[8] A. Garriga-Alonso, C. E. Rasmussen, and L. Aitchison. "Deep convolutional networks as shallow gaussian processes." In: *arXiv preprint arXiv:1808.05587* (2018).

[9] R. B. Gramacy. *Bayesian treed Gaussian process models*. University of California, Santa Cruz, 2005.

[10] R. M. Gray. *Entropy and information theory*. Springer Science & Business Media, 2011.

[11] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. "Hypercolumns for object segmentation and fine-grained localization." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 447–456.

[12] J. A. Hartigan. *Clustering algorithms*. John Wiley & Sons, Inc., 1975.

[13] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[14] D. Hernández-Lobato, J. Hernández-lobato, and P. Dupont. "Robust multi-class Gaussian process classification." In: *Advances in neural information processing systems* 24 (2011).

[15] N. J. Higham. *Accuracy and stability of numerical algorithms*. SIAM, 2002.

[16] D. S. Hochbaum. "An efficient algorithm for image segmentation, Markov random fields and related problems." In: *Journal of the ACM (JACM)* 48.4 (2001), pp. 686–701.

[17] M. Hossin and M. N. Sulaiman. "A review on evaluation metrics for data classification evaluations." In: *International journal of data mining & knowledge management process* 5.2 (2015), p. 1.

[18] L. Hyafil and R. L. Rivest. "Constructing optimal binary decision trees is NP-complete." In: *Information Processing Letters* 5.1 (1976), pp. 15–17. ISSN: 0020-0190. DOI: https://doi.org/10.1016/0020-0190(76)90095-8.

[19] C. KI Williams. *Gaussian processes for machine learning*. Taylor & Francis Group, 2006.

[20] A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollár. "Panoptic segmentation." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 9404–9413.

[21] V. Kotu and B. Deshpande. *Data science: concepts and practice*. Morgan Kaufmann, 2018.

[22] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of massive data sets*. Cambridge university press, 2020.

[23] X. Liu, Z. Deng, and Y. Yang. "Recent progress in semantic image segmentation." In: *Artificial Intelligence Review* 52.2 (2019), pp. 1089–1106.

[24] LukeTonin. *Luketonin/simple-deep-learning: Simple data and simple models to learn the fundamentals of deep learning.*

[25] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos. "Image Segmentation Using Deep Learning: A Survey." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.7 (2022), pp. 3523–3542. DOI: 10.1109/TPAMI.2021.3059968.

[26] T. P. Minka and R. Picard. "A Family of Algorithms for Approximate Bayesian Inference." AAI0803033. PhD thesis. USA, 2001.

[27] K. P. Murphy. *Machine learning : a probabilistic perspective*. Cambridge, Mass. [u.a.]: MIT Press, 2013. ISBN: 9780262018029 0262018020.

[28] R. Okuta, Y. Unno, D. Nishino, S. Hido, and C. Loomis. "CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations." In: *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*. 2017.

[29] J.-O. Palacio-Niño and F. Berzal. "Evaluation metrics for unsupervised learning algorithms." In: *arXiv preprint arXiv:1905.05667* (2019).

[30] M. Parniske. "Arbuscular mycorrhiza: the mother of plant root endosymbioses." In: *Nature Reviews Microbiology* 6.10 (2008), pp. 763–775. ISSN: 1740-1534. DOI: 10.1038/nrmicro1987.

[31] T. Parr, T. Lapusan, and P. Grover. *Parrt/dtreeviz: A python library for decision tree visualization and model interpretation.*

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[33] J. Quinonero-Candela and C. E. Rasmussen. "A unifying view of sparse approximate Gaussian process regression." In: *The Journal of Machine Learning Research* 6 (2005), pp. 1939–1959.

[34] L. Raileanu and K. Stoffel. "Theoretical Comparison between the Gini Index and Information Gain Criteria." In: Jan. 2002.

[35] P. K. Sahoo, S. Soltani, and A. K. Wong. "A survey of thresholding techniques." In: *Computer vision, graphics, and image processing* 41.2 (1988), pp. 233–260.

[36] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[37] J. Watter. "Light Microscopy Image Analysis using Neural Networks." Master's Thesis. Technical University of Munich, Apr. 2021.

[38] C. Williams and D. Barber. "Bayesian classification with Gaussian processes." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.12 (1998), pp. 1342–1351. DOI: 10.1109/34.735807.

A. Appendix

A.1. Further visual comparisons of the extended MNIST data set

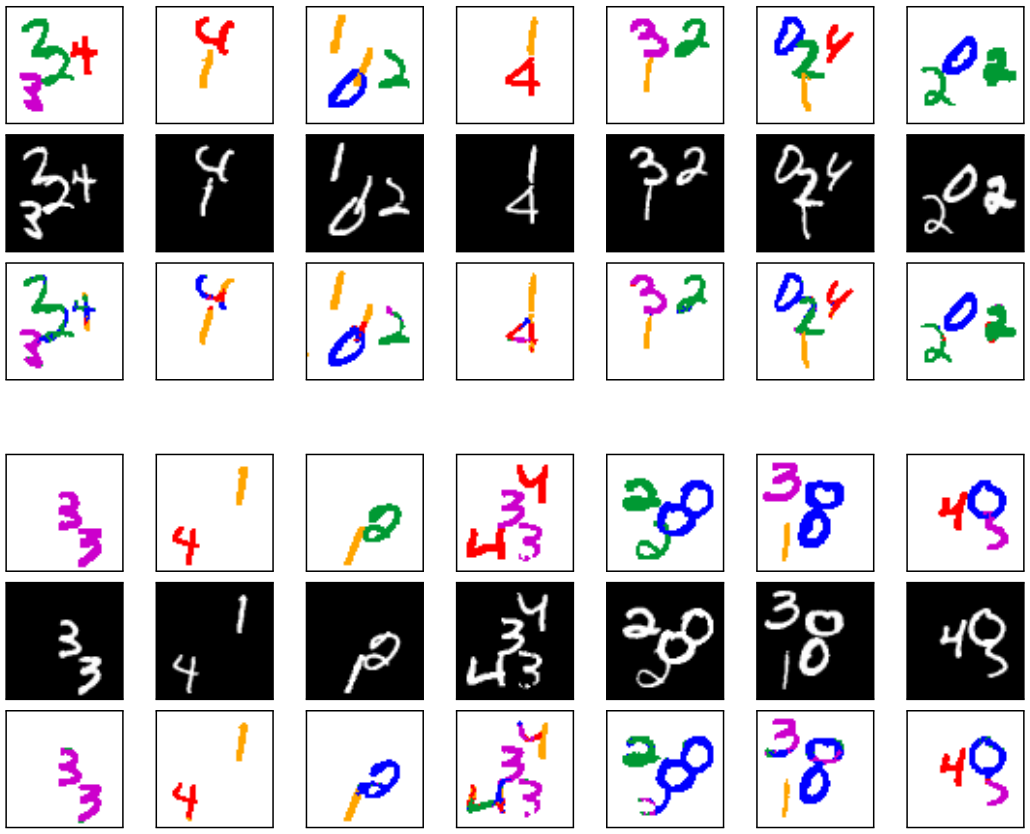


Figure A.1.: Comparison of the output of the treed Gaussian process trained on the extended MNIST dataset. Top: groundtruth, middle: original, bottom: model output.

A.2. Discussion using PCA for the decision tree

We trained a treed Gaussian process on the extended MNIST dataset. We used an 80/20 training-test split. In the figures A.2 and A.3, we can see that for more images used to train/validate the model, the more insignificant the difference. The difference slightly oscillates at the zero line. After 70 images, the difference is neglectable.

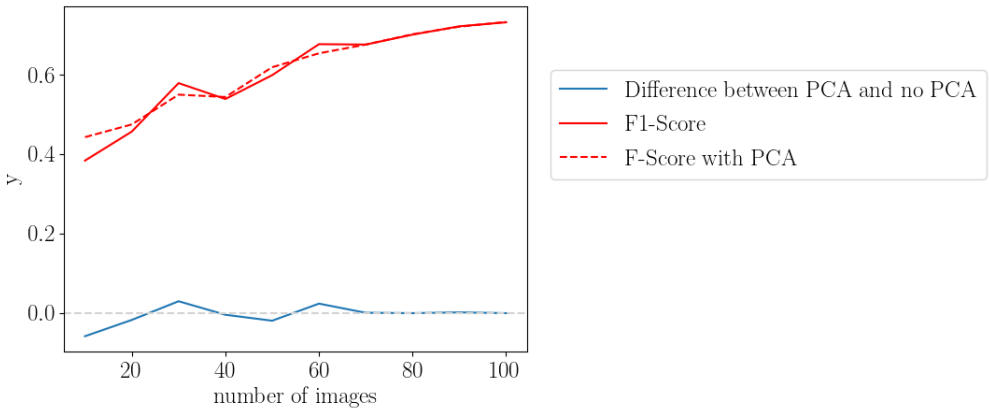


Figure A.2.: Difference of PCA and no PCA using the mean F1-Score.

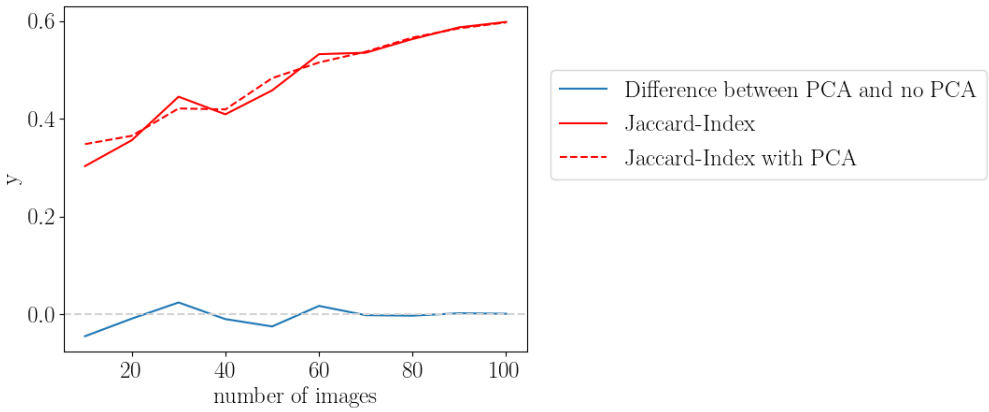


Figure A.3.: Difference of PCA and no PCA using the mean Jaccard-Index.

A.3. Further visual comparisons of the light microscopy images

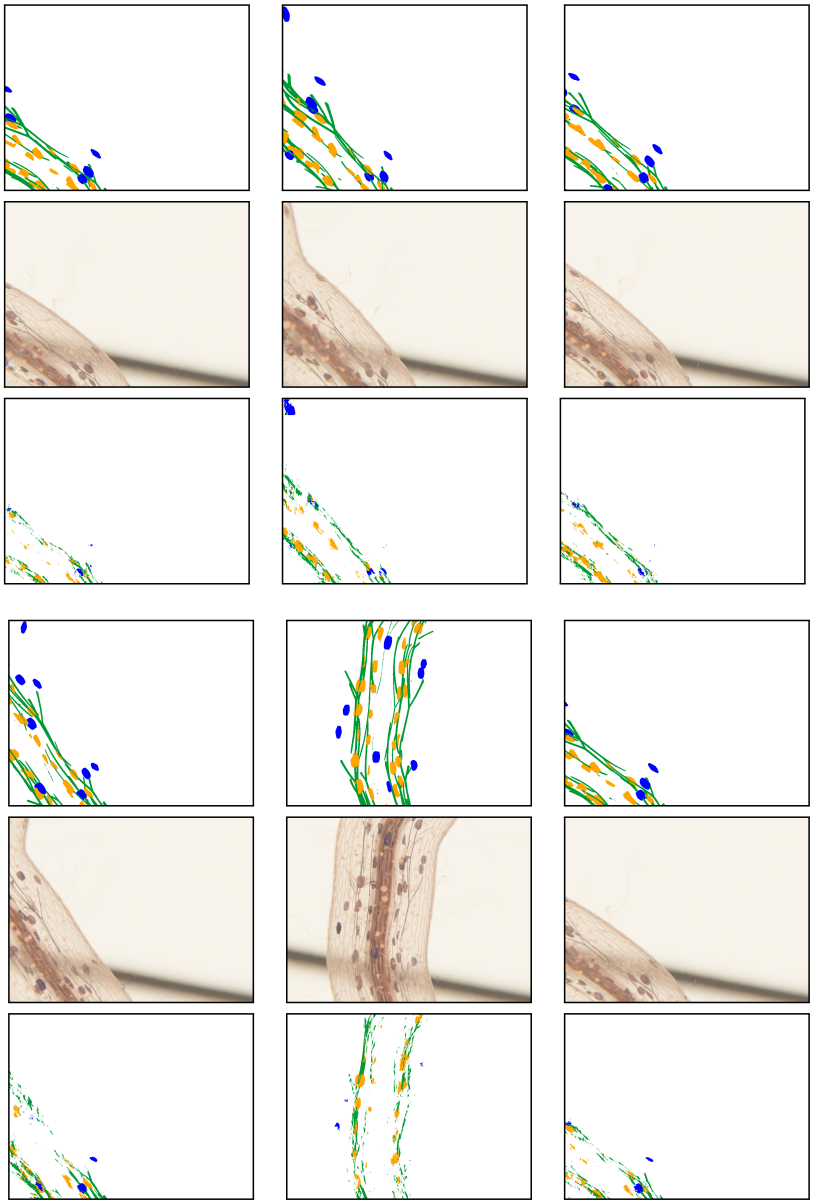


Figure A.4.: Comparison of the output of the treed Gaussian process trained on the light microscopy image dataset. Top: groundtruth, middle: original, bottom: model output.

List of Figures

List of Tables