

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Bayesian Active Learning for Microscopy Image Segmentation

verfasst von | submitted by
Jakob Bleiberschnig

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of
Master of Science (MSc)

Wien | Vienna, 2025

Studienkennzahl lt. Studienblatt | Degree
programme code as it appears on the
student record sheet:

UA 066 910

Studienrichtung lt. Studienblatt | Degree
programme as it appears on the student
record sheet:

Masterstudium Computational Science

Betreut von | Supervisor:

Assoz. Prof. Dipl.-Ing. Dr.techn. Sebastian
Tschitschek BSc

Acknowledgement

First and foremost, I would like to express my gratitude to my supervisor, Prof. Sebastian Tschatschek, for his invaluable input through numerous discussions and his detailed and helpful feedback, which were crucial to the development of this thesis. His guidance and insights have been instrumental in shaping the direction of my research. I am deeply grateful to Richard Paul for his generous support and time, guiding me through the intricacies of Bayesian statistics and instance segmentation. His assistance in helping me identify a compelling topic and his proofreading were of immense help. My sincere thanks also go to Dr. Katharina Nöh and Johannes Seiffart for their thoughtful feedback and the engaging discussions we shared, which enriched my understanding and contributed greatly to my work. Finally, I would like to extend my gratitude to Dr. Hanno Scharf for providing me with the opportunity to collaborate with Forschungszentrum Jülich, which allowed me to apply my research in a meaningful context.

Thank you all for your support and contributions, without which this thesis would not have been possible.

List of Figures

1	Semantic and instance segmentation of the same input image.	2
2	Microscopy image transformation as neighbor and cell distance maps.	15
3	Transformations to compute the neighbor distance.	15
4	Image with and without checkerboard artifacts.	16
5	Segmentation performance of Omnipose compared with Cellpose and a combination.	18
6	Jaccard Index as a function of IOU threshold for different datasets.	19
7	High-level overview of the methodology.	19
8	Conceptual overview of the architecture and post-processing step.	20
9	Example images for each selected dataset.	29
10	Distribution of cell count per image and cell area measured in pixels.	30
11	Histograms of the pixel values of background pixels and edge proportions.	31
12	Segmentation mask generated by applying post-processing with Otsu thresholds onto the ground truth cell distances compared to the ground truth segmentation mask.	34
13	Selecting the post-processing thresholds with Otsu's method vs. via grid search.	34
14	Comparison of the segmentation performance of different network architectures.	35
15	Segmentation performance of four models pretrained on each dataset and evaluated on all other datasets.	37
16	Segmentation performance of four models pretrained on each dataset and fine-tuned on all other datasets.	37
17	Pretraining - fine-tuning dataset combinations	38
18	F1-score of each individual test set image from HELA	39
19	F1-score of each individual test set image from TOIAM	39
20	Segmentation performance on the test set after fine-tuning the pretrained models.	40
21	STD acquisition scores on the pool set of TOIAM	41
22	Four most uncertain images of the MSEG pool set.	42
23	Four most uncertain images of the HELA pool set.	43
24	STD acquisition score on the pool set of HELA.	43
25	Four most uncertain images of the CPOSE pool set.	44
26	Prediction and uncertainty map with and without checkerboard artifacts.	45

List of Tables

1	Impact of the post-processing parameters th_{seed} and th_{cell} on the instance segmentation result.	15
2	Results of CNN based distance predictions on different datasets.	18
3	Detailed overview of the used neural network architecture.	21
4	Overview of active learning settings during fine-tuning.	26
5	Number of fine-tuning training images used for each dataset.	26
6	Chosen hyperparameters during fine-tuning of the pretrained models.	26
7	Hyperparameters used for pretraining models on the four datasets.	27
8	Properties of the used datasets.	28
9	Qualitative properties of the datasets.	31
10	Overview of the most important used libraries and their versions.	32
11	Optimized hyperparameters for watershed post-processing for each dataset. . . .	33
12	Architectural changes to reduce checkerboard artifacts.	45

Abstract

Data efficient learning algorithms have been the key to enable machine learning applications in domains where data is scarce. An example of such a domain are microscopy images, since in many cases the generation of such images and especially the annotation is costly. This thesis focuses on instance segmentation whereby a model learns from training data to identify individual objects on a microscopy image. The creation of manually annotated training images is time consuming and needs to be done by experts in the respective field. Thus, our goal is to reduce the number of training images as much as possible. To achieve this goal, we train the model by using active learning. Through active learning, the model is enabled to decide on its own, which images should be annotated next and used for training. By this we want the model to only learn from training images with a maximum level of information, instead of wasting annotation effort on images that are less informative. We utilize a neural network with a U-Net architecture as the base model for segmenting the microscopic images. A standard neural network learns a single point estimate of the network parameters for a given dataset, which can have low performance if the training set does not represent the underlying data distribution well. This can easily be the case if we have a small training set. In contrast to this, Bayesian neural networks learn not a point estimate, but a distribution over the parameters. This further allows to predict a distribution of possible outputs for a single input image. A Bayesian neural network can thus be seen as an ensemble of infinitely many neural networks. They are a very powerful tool for learning the uncertainty caused by limited data, and therefore capable of estimating the informativeness of the input images.

Our main contribution is to actively sample the training images by utilizing the uncertainty estimation generated by a Gaussian Bayesian neural network as a measure of informativeness. This work is one of the first studies that apply this Bayesian active learning approach to instance segmentation of microscopy images. Gaussian Bayesian neural networks are more expensive to train than other common Bayesian based methods and require an additional hyperparameter, but come with the advantage of a more complex parameter distribution that can lead to a more detailed estimation of the uncertainty.

We conduct experiments on four different microscopy image datasets, where we compare active learning with uniform sampling as a baseline. Additionally we pretrain each U-Net on another dataset in a transfer learning manner. We find that our chosen active learning framework can outperform uniform sampling and thus reduce the amount of training images for two of our four used datasets. A limitation of this work is, that the annotation costs are considered to be constant for all images. Future research could extend this work in that matter.

Zusammenfassung

Daten-effiziente Lernalgorithmen sind der Schlüssel, um maschinelles Lernen in Bereichen zu ermöglichen, in denen wenig Daten vorhanden sind. Ein Beispiel für einen solchen Bereich sind Mikroskopiebilder, da die Erstellung und insbesondere die Annotation solcher Bilder oft kostspielig ist. Diese Abschlussarbeit behandelt die Anwendung der Instanz-Segmentierung, bei der ein Modell lernt aus annotierten Trainingsdaten einzelne Objekte auf einem Mikroskopiebild zu identifizieren. Die Erstellung manuell annotierter Trainingsbilder ist zeitaufwändig und muss von Experten im jeweiligen Fachgebiet durchgeführt werden. Daher ist es unser Ziel, die Anzahl der benötigten Trainingsbilder so weit wie möglich zu reduzieren. Um dieses Ziel zu erreichen, trainieren wir das Modell durch Active-Learning. Dadurch wird das Modell in die Lage versetzt, selbst zu entscheiden, welche Bilder als nächstes annotiert und für das Training verwendet werden sollen. Auf diese Weise soll das Modell nur aus Trainingsbildern mit einem maximalen Informationsgehalt lernen, anstatt Ressourcen für die Annotation weniger informativer Bilder zu verschwenden. Wir nutzen ein neuronales Netzwerk mit einer U-Net-Architektur als Basismodell zur Segmentierung der Mikroskopiebilder. Ein Standard-neuronales Netzwerk lernt eine Punktschätzung der Netzwerkparameter für einen gegebenen Datensatz, was zu niedriger Genauigkeit führen kann, wenn der Trainingsdatensatz die zugrunde liegende Datenverteilung nicht gut repräsentiert. Das kann leicht der Fall sein, wenn wir einen kleinen Trainingsdatensatz haben. Im Gegensatz dazu lernen Bayessche neuronale Netzwerke keine Punktschätzung, sondern eine Verteilung über die Parameter. Dies ermöglicht es, eine Verteilung über mögliche Ausgaben für ein einzelnes Eingabebild vorherzusagen. Ein Bayessches neuronales Netzwerk kann daher als Ensemble unendlich vieler neuronaler Netzwerke betrachtet werden, bei denen jedes Netzwerk eine leicht unterschiedliche Lösung für ein gegebenes Eingabebild erzeugt. Diese Netzwerke sind ein sehr leistungsfähiges Werkzeug, um von Unsicherheiten in den Trainingsdaten zu lernen und dadurch den Informationsgehalt der Eingabebilder abzuschätzen.

Unser Hauptbeitrag besteht darin, die Trainingsbilder aktiv zu sampeln, indem wir die durch ein Gaußsches Bayessches neuronales Netzwerk generierte Unsicherheitsschätzung als Maß für den Informationsgehalt eines Bildes nutzen. Diese Arbeit ist eine der ersten Studien, die diesen Bayesschen Active-Learning Ansatz auf die Instanz-Segmentierung von Mikroskopiebildern anwendet. Gaußsche Bayessche neuronale Netzwerke sind teurer im Training als andere gängige Bayessche Methoden und erfordern einen zusätzlichen Hyperparameter, bieten jedoch den Vorteil einer komplexeren Parameterverteilung, die zu einer detaillierteren Schätzung der Unsicherheit führen kann.

Es wurden Experimente mit vier verschiedenen Mikroskopie-Datensätzen durchgeführt, bei denen Active-Learning mit Uniform-Sampling verglichen wird. Zusätzlich wird der Effekt untersucht, der sich ergibt, wenn das U-Net auf einem anderen Datensatz vortrainiert wird. Es konnte festgestellt werden, dass das gewählte Active-Learning-Framework in zwei von vier verwendeten Datensätzen Uniform-Sampling übertrifft. Eine Limitierung dieser Arbeit ist, dass die Annotationskosten für alle Bilder als konstant betrachtet werden. Zukünftige Studien könnten diese Arbeit dahingehend erweitern.

Table of Contents

1	Introduction	2
2	Background	4
2.1	Bayesian Neural Networks	4
2.2	Active Learning Theory	8
2.3	Transfer Learning	12
2.4	Neural Network Architectures for Microscopy Image Segmentation	13
2.5	Checkerboard Artifacts	16
3	Related Work	16
3.1	Related Active Learning Literature	16
3.2	Related Instance Segmentation Literature	17
4	Methodology	18
4.1	Model Architecture and Post-Processing Algorithm	19
4.2	Active Learning Framework	20
4.3	Evaluation Metrics	26
5	Experimental Setup	27
5.1	Used Data	27
5.2	Development Environment	30
6	Results	32
6.1	Post-Processing Hyperparameter Selection	33
6.2	Neural Network Architecture Comparison	35
6.3	Pretraining Dataset Selection	35
6.4	Dependency of the Segmentation Performance on the Cell Count	36
6.5	Active Learning Results	38
6.6	Analysis of Checkerboard Artifacts	44
7	Discussion	45
8	Conclusion	46
	References	48
	A Propositions and Proofs	54

1 Introduction

Microorganisms are omnipresent and play crucial roles in numerous natural processes as well as industrial production, such as insulin production by *Escherichia coli* [1]. Since many decades biotechnological industries are aware of the importance of understanding and analyzing the mechanisms of growth and productivity of such microorganisms. As [1] further explains, conventional analyses of microorganisms historically assumed that isogenetic bacterial populations exhibit high local homogeneity, with only minor cell-to-cell variations. However, research over the past two decades has revealed substantial differences among individual cells within the same isogenetic bacterial colony, including variations in growth, stress resistance, and other properties. [2] highlights, that recent progress in lab-on-a-chip technology has significantly enhanced experimental methods, enabling the observation of individual cell development with high spatial and temporal precision. The resulting data that is collected from such experiments are large time-lapse image sequences of the growing cell populations. The ability to produce such extensive datasets underlines the importance of automated image analysis methods.

One type of information that we are interested in are time-resolved cellular properties, such as cell size, over the lifespan of a single cell. Therefore, it is necessary to track the same cell in every image of the time-lapse sequence. If the population grows through cellular division, the result of such a cell-tracking analysis can then be displayed in a lineage tree, showing the cellular divisions and resulting child cells over time. [2]

An essential step to track individual cells over a sequence of time-lapse images is the automated segmentation of the individual organisms that are present in these images, i.e., instance segmentation regarding the individual cells. Instance segmentation is a labeling task, where each pixel in an image is labeled with an index that represents an object of a predefined class. It is distinguished from semantic segmentation where each pixel gets assigned to a class label and distinguishing different objects of the same class is not possible. See Figure 1 as an example. Instance segmentation on microscopy images of microorganisms is the task of recognizing each individual cell as a separate object belonging to one species. [3]

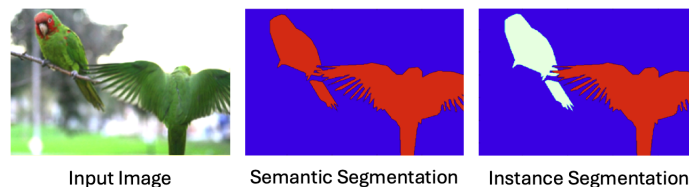


Figure 1: Semantic segmentation distinguishes between background and object class(es), while instance segmentation distinguishes between background and individual objects. Image is taken from [4].

Typically, instance segmentation is performed using a convolutional neural network and thus is in demand of large amounts of training images annotated by human experts. High-quality microscopy images are not only rare and expensive to produce, but the expert annotation process is also time-consuming and costly. One way to reduce the amount of training data and, therefore, reducing the cost of training is active learning. [5]

The goal of this work is to reduce the amount of training data that is needed for the automated instance segmentation of microscopy images. To be more specific, the central goal of this thesis is to tackle microscopy instance segmentation using an active learning approach that leverages the uncertainty estimation of Bayesian neural networks (BNN) to sample the most informative training images for labeling. This should lead to an increase in model performance while using less labeled training data than comparable methods. This goal is formulated in the following high-level research question.

Can the uncertainty estimation of a Gaussian Bayesian neural network be used to successfully facilitate active learning for instance segmentation?

In previous work similar goals have been pursued with different approaches to obtain an estimation of uncertainty as a measure for informativeness. One of these approaches is to use the Monte Carlo dropout method described by [6] to create a BNN with a Bernoulli distribution as the posterior. This attempt has been applied to microscopy instance segmentation by [7]. This comes with the disadvantage of approximating the complex weight distribution with a very simple and discontinuous Bernoulli distribution. Important characteristics of the uncertainty might be lost through this simplification. Another approach to estimate uncertainty is to utilize the disagreement of an ensemble of models [8]. This comes with the downside of requiring to train an increasing number of neural networks to increase the accuracy of the uncertainty estimation. Another approach is to let a standard neural network output a probability vector that represents the probabilities of the input belonging to each possible class. The uncertainty can then be computed via the Shannon entropy of this vector. As pointed out by [9], this uncertainty estimation cannot capture epistemic uncertainty, which is the uncertainty that results from a lack of knowledge about the underlying process that generates the data. During active learning however, we want to measure and reduce the epistemic uncertainty and hence this approach to estimate uncertainty is not recommended. Gaussian Bayesian neural networks are one way to tackle the disadvantages of the above described approaches for uncertainty estimation, which is why this method is selected for this work.

The contribution of this thesis is to utilize the uncertainty estimation of diagonally Gaussian BNNs. In such neural networks, the uncertainty distribution of the weights (called posterior) is a Gaussian distribution with a diagonal covariance matrix. The covariance matrix and the mean of the Gaussian distribution are optimized in the training process. The diagonal Gaussian distribution is a more complex approximation to the true posterior than, for example, a Bernoulli distribution that is used in [7]. Therefore a diagonally Gaussian BNN leads to a more complex uncertainty representation, with the chance of estimating the uncertainty in the training data more accurately. This work uses a learning algorithm for training diagonally Gaussian BNNs, that was published by [10]. So far, this has never been applied to an active learning framework for instance segmentation for microscopy images.

With this knowledge the research question from above can be stated in more detail as follows.

Can the Bayesian uncertainty estimation from a diagonally Gaussian Bayesian neural network be used for selecting training data to train a model in an active learning manner, resulting in a better model performance compared to sampling training data uniformly at random?

We found that the chosen active learning framework in combination with the uncertainty estimation from a Gaussian BNN can outperform random sampling on two out of four microscopy image datasets. All four datasets contain images that are created from microbiological experiments. The datasets vary in homogeneity between the images and dataset size. The results suggest that our method worked best on the smallest and the most homogeneous dataset.

This thesis is structured as follows. Section 2 starts with an overview of Bayesian neural networks. Next, we review the theory of active learning and discuss different active learning scenarios, sampling strategies and acquisition functions. We further discuss different neural network architectures that are widely used in segmenting microscopy images and end the chapter with an introduction to checkerboard artifacts. In section 3 we analyze two different microscopy segmentation studies and one work that also uses Bayesian neural networks in an active learning framework. In section 4 the chosen neural network architecture, the active learning method and the evaluation metrics are described and reasons are given for each choice. Section 5 describes the four chosen datasets and which development environment (hardware and software) was used. The results of all conducted experiments are presented and analyzed in section 6. We discuss the results in section 7 and relate them to other results from the literature. In section 8, we summarize the main outcomes of this thesis and give suggestions for future work.

2 Background

2.1 Bayesian Neural Networks

Classical neural networks often suffer from the following common problems. One of which is overfitting, another one is that standard neural networks are often very confident in their decisions even if the underlying data does not give rise to such a level of confidence [10]. The first problem is usually tackled by regularization techniques such as dropout or weight decay [11]. The second problem is, in case of classification, often tackled by letting the standard neural network generate a probability vector where the uncertainty can then be measured by, for example, the Shannon entropy [12]. This way of computing uncertainty can only measure aleatoric uncertainty. This type of uncertainty arises from the inherent randomness of the process that generates the data. Therefore aleatoric uncertainty cannot be reduced by collecting more data-points. In contrast to this there is epistemic uncertainty. This uncertainty results from the lack of knowledge about the data generating process and can be explained and reduced as more data becomes available. When performing active learning we are highly interested in reducing the uncertainty by collecting and learning from new data-points, thus we are interested in modeling the epistemic uncertainty.[9]

The above two problems can also be tackled by a Bayesian inference approach applied to neural networks resulting in a Bayesian neural network (BNN). For describing BNNs, this work uses the same notation as in [10]. We start with a neural network, that receives an input $x \in \mathcal{X} \subseteq \mathbb{R}^d$ and instead of producing the output $y \in \mathcal{Y} \subseteq \mathbb{R}^q$ directly, it assigns a probability to each element in the output space. So the network can be viewed as a probabilistic model $P(y|x, w)$ with w being the parameters of the model. Bayesian inference offers a way to utilize prior knowledge of the network parameters w in form of the prior distribution $P(w)$ to infer a posterior probability over the parameters $P(w|D)$ with $D = \{(x, y), x \in \mathcal{X}, y \in \mathcal{Y}\}$ being the data that is available to the model. This relationship is described by Bayes' theorem.

$$P(w|D) = \frac{P(D|w)P(w)}{P(D)} \rightarrow \text{Posterior} = \frac{\text{Likelihood} \cdot \text{Prior}}{\text{Evidence}} \quad (1)$$

The individual terms are elaborated in the following.

- **Likelihood.** The probability that the data D can be explained by the model given the parameters w is referred to as likelihood. In a neural network we typically choose the loss function $L_D(w)$ with given parameters w to be proportional to the negative log-likelihood. In other words we have:

$$P(D|w) = \frac{\exp(-L_D(w))}{Z_L} \quad (2)$$

with $Z_L = \int \exp(-L_D(w))$ being a normalizing factor. For the choice of L2-loss $L_D(w) = L_{D,L2} = \frac{1}{2} \sum \|y^* - y\|_2^2$ with y^* being the exact solution and y being the prediction of the neural network with parameters w , the likelihood corresponds to a Gaussian distribution.[13]

- **Prior.** The prior is the knowledge about the weight distribution that is (subjectively) assumed without knowing the data. It is difficult to make a specific assumption about the weight distribution of a neural network, which is why it is often assumed that the weights are independently normally distributed.

$$P(w) = N(0, \sigma^2) = \frac{\exp(-\alpha \|w\|_2^2)}{Z_N} \quad (3)$$

with $\alpha = \frac{1}{2\sigma^2}$ and Z_N being the normalization factor. So the parameter α controls the variance of the Gaussian prior. [13]

- **Posterior.** Given Bayes theorem in Equation 1, the probability of the weights given the data can be expressed from the likelihood and the prior as

$$P(w|D) = \frac{\exp(-L_D(w) - \alpha\|w\|_2^2)}{Z} \quad (4)$$

[13]

Finding the most probable (maximum a posteriori) weight w_{map} from $P(w|D)$ is equivalent to finding the minimizing w for the negative exponent. This is nothing else than the chosen L2-loss plus a regularization term.

$$w_{map} = \arg \min_w L_D(w) + \alpha\|w\|_2^2$$

This shows how BNNs naturally introduce regularization by assuming a prior distribution to the weights. In case of a Gaussian prior, this regularization is equal to weight decay. [10]

Note that computing or integrating over the posterior distribution is computationally infeasible for deep neural networks, since the number of parameters in w can be extremely large and the mathematical form of a neural network does not allow exact integration. [10]

- **Evidence.** The evidence is the normalizing factor for the posterior and thus constant in w . Given the above likelihood and prior, the evidence is

$$P(D) = Z = \int \exp(-L_D(w) - \alpha\|w\|_2^2) dw$$

[13].

If we assume the data D to be independently and identically distributed (i.i.d.), we can state the likelihood of the dataset D as the product over all data-point-wise likelihoods $P(y|x, w)$.

$$P(D|w) = \prod_{i=1}^m P(y^{(i)}|x^{(i)}, w), \text{ with } (x^{(i)}, y^{(i)}) \sim D$$

As mentioned before, the above likelihood becomes Gaussian for the choice of L2-loss. Therefore, under the i.i.d. assumption, the data-point-wise likelihood $P(y|x, w)$ becomes Gaussian as well and is defined through the mean $\mu \in \mathbb{R}^q$ and the covariance matrix $\Sigma \in \mathbb{R}^{q \times q}$. If we now take the expectation value of the data-point-wise likelihood with respect to w , we get the so called predictive distribution [14].

$$P(y|x) = \mathbb{E}_{P(w|D)}[P(y|x, w)] = \int P(y|x, w) \cdot P(w|D) dw \quad (5)$$

The predictive distribution $P(y|x)$ represents the distribution of y over the solution space $\mathcal{Y}$ for a given input x independent of w . In other words, the above described BNN maps each input $x \in \mathcal{X}$ onto the parameters of a probability distribution on $\mathcal{Y}$. From this it is easy to see that we can compute the uncertainty in the prediction of a single input x from a BNN by computing the variance of $P(y|x)$. Other uncertainty measures exist and are discussed in section 2.2. [9] suggest that by using the right uncertainty measure, BNNs are able to capture epistemic uncertainty and leave aside aleatoric uncertainty.

Variants of Variational Bayesian Inference

Now the question arises, how we can compute or at least approximate the predictive distribution $P(y|x)$. One way is to approximate the integral in Equation 5 by Monte Carlo (MC) integration. The problem here is that we cannot sample directly from the posterior distribution in Equation 4.

There are several methods to overcome this problem such as variants of Markov Chain Monte Carlo (MCMC) methods or variational Bayesian inference (VI) [15]. While MCMC methods can produce samples from the exact posterior, it can take very long to reach this state in practice [15]. VI on the contrary, is typically computationally less expensive, which is especially relevant for deep neural networks with a high-dimensional parameter space [16]. This advantage is also the reason why we chose VI for this work.

By VI we do not aim to sample from the exact posterior distribution, but try to find a good approximation to the posterior. We denote this approximate posterior as the distribution $q(w|\theta)$ over w given a set of variational parameters θ . This distribution should have a known functional form that allows to draw samples efficiently. This property is then used to efficiently perform MC integration.

There are two common choices in literature for the functional form of the approximate posterior. One is the Bernoulli distribution, which leads to Bernoulli Bayesian neural networks. The other one is the diagonal Gaussian distribution, which leads to diagonally Gaussian neural networks. In case of a Bernoulli BNN, we define the approximate posterior $q(w^{(l)}|\theta^{(l)})$ for the weight vector $w^{(l)}$ of the l -th neural network layer as

$$w^{(l)} = \theta^{(l)} \cdot \text{diag}(\alpha^{(l)}),$$

where the elements of the binary vector $\alpha^{(l)}$ are independently sampled from a Bernoulli distribution

$$\alpha_i^{(l)} \sim \text{Bernoulli}(p^{(l)}) \text{ for } i = 1, \dots, k_{l-1}.$$

Here $\text{diag}(\alpha^{(l)})$ is a diagonal matrix with the elements of the vector $\alpha^{(l)}$ on its main diagonal. The parameters $\theta^{(l)}$ are the variational parameters θ of each layer l that need to be optimized. The probability of the Bernoulli distribution $p^{(l)}$ is a hyperparameter and typically set to be constant for each layer. The length k_{l-1} of the vector $\alpha^{(l)}$ is given by the dimensionality of the output of the previous layer $l - 1$. [6] In short, in a Bernoulli BNN, the weights are individually set to zero with probability $1 - p^{(l)}$. In [6] it is also shown that using a Bernoulli BNN does not introduce additional parameters compared to a deterministic network of the same size.

In contrast to that, diagonally Gaussian BNNs have a Gaussian approximate posterior with mean vector μ and a diagonal covariance matrix Σ [17]. This means, we have a multivariate Gaussian distribution over all weights with zero correlation between the weights. Thus, we can express the approximate posterior for each element $w_i \in \mathbb{R}$ in the weight vector w as an independent univariate Gaussian with mean $\mu_i \in \mathbb{R}$ and variance $\sigma_i \in \mathbb{R}_+$. Intuitively we could view the mean vector μ and variances Σ to form the variational parameters θ of the approximate posterior. The problem here is, that during the training of a neural network, the network parameters can easily become negative. However, the variance must always be a positive value. This is why [10] suggest to formulate the parameters θ of a diagonal Gaussian BNN as $\theta = \{w_i = (\mu_i, \rho_i), i = 1, 2, \dots\}$ and define the variance as $\sigma_i = \log(1 + \exp(\rho_i))$. This ensures that the variance σ_i is positive for any value $\rho_i \in \mathbb{R}$. It is easy to see that a diagonal Gaussian BNN doubles the number of parameters in the network, since each weight w_i of a deterministic NN is represented by a mean μ_i and a variance σ_i .

In this work we only consider diagonally Gaussian BNNs, which is why in the following we always mean Gaussian BNN when we speak of BNN. The reason for the choice of a Gaussian BNN over a Bernoulli BNN is, that by allowing the posterior approximation to be more complex (we have two learnable parameters in Gaussian BNNs and only one learnable parameter in Bernoulli BNNs for each weight), we may find a better approximation to the true posterior and therefore find a better uncertainty estimation of the networks prediction.

Gaussian Posterior Approximation via BayesByBackprop

To find a good approximation of the posterior, we can use the Kullback-Leibler (KL) divergence. It is a measure for how much two distributions differ from each other. In this context finding a

good approximate posterior $q(w|\theta)$ means finding a value θ^* that minimizes the KL-divergence between the approximate posterior and the true posterior.

$$\theta^* = \arg \min_{\theta} \text{KL} [q(w|\theta) || P(w|D)]$$

The KL-divergence can be reformulated as the variational free energy (VFE) $\mathcal{F}(D, \theta)$ [10]. The proof is shown in section A. Thus, we can find the best approximate posterior by minimizing the VFE. [17]

$$\theta^* = \arg \min_{\theta} \mathcal{F}(\mathcal{D}, \theta)$$

with the VFE given as

$$\mathcal{F}(\mathcal{D}, \theta) = \text{KL} [q(w|\theta) || P(w)] - \mathbb{E}_{q(w|\theta)} [\log P(\mathcal{D}|w)]$$

As elaborated in [17], the VFE consists of a prior dependent term ($\text{KL} [q(w|\theta) || P(w)]$) and a data dependent term ($-\mathbb{E}_{q(w|\theta)} [\log P(\mathcal{D}|w)]$). Intuitively, the prior dependent term regularizes the variance of $q(w|\theta)$ to be close to the variance of the prior. This term is also called the *complexity cost*. The data dependent term decreases when the predictive accuracy of the network increases. It is called the *error cost*. The regularization can be controlled by multiplying a factor to the prior dependent term [10]. In this work this factor will be referred to as *KL-weight*.

[10] introduced a new algorithm, called *Bayes by Backprop* which aims to solve variational Bayesian inference with a diagonal Gaussian posterior approximation by finding an approximate solution for the value of θ^* . To achieve this, in a first step the VFE is approximated by

$$\begin{aligned} \mathcal{F}(\mathcal{D}, \theta) &\approx \sum_{i=1}^n \log q(w^{(i)}|\theta) - \log P(w^{(i)}) - \log P(D|w^{(i)}) \\ &= \sum_{i=1}^n f(w^{(i)}, \theta), \end{aligned}$$

where $w^{(i)}$ are Monte Carlo samples from $q(w|\theta)$ and n is the chosen number of MC samples. Here $f(w^{(i)}, \theta)$ represents the VFE component of a single MC sample. An intuition, why this approximation is a good choice is given in section A. In a second step the proposition also given in section A is applied to the approximated VFE. This proposition allows to express the derivative of an expectation value as an expectation value of a derivative. To show this, we first express the approximation of $\mathcal{F}(\mathcal{D}, \theta)$ as an expectation value.

$$\mathcal{F}(\mathcal{D}, \theta) \approx \sum_{i=1}^n f(w^{(i)}, \theta) = \mathbb{E}_{q(w|\theta)} [f(w^{(i)}, \theta)], \text{ for } n \rightarrow \infty$$

We can now apply the proposition from section A to calculate the gradients with respect to the mean μ and to ρ as follows.

$$\begin{aligned} \frac{\partial}{\partial \mu} \mathbb{E}_{q(w|\theta)} [f(w^{(i)}, \theta)] &= \mathbb{E}_{q(\epsilon)} \left[\frac{\partial f(w^{(i)}, \theta)}{\partial w} \frac{\partial w}{\partial \mu} + \frac{\partial f(w^{(i)}, \theta)}{\partial \mu} \right] \\ \frac{\partial}{\partial \rho} \mathbb{E}_{q(w|\theta)} [f(w^{(i)}, \theta)] &= \mathbb{E}_{q(\epsilon)} \left[\frac{\partial f(w^{(i)}, \theta)}{\partial w} \frac{\partial w}{\partial \rho} + \frac{\partial f(w^{(i)}, \theta)}{\partial \rho} \right] \end{aligned}$$

The distribution $q(\epsilon)$ is a standard normal distribution that we can use to express a sample weight as $w^{(i)} = \mu + \log(1 + \exp(\rho)) * \epsilon$ with $\epsilon \sim N(0, 1) = q(\epsilon)$. Note that with this we have the derivatives $\frac{dw}{d\mu} = 1$ and $\frac{dw}{d\rho} = \frac{\epsilon}{1 + \exp(-\rho)}$. Also note that the term $\frac{\partial f(w^{(i)}, \theta)}{\partial w}$ is simply

the gradient of a deterministic neural network with the parameters $w^{(i)}$. Calculating the above expectation value for each weight update of the neural network would be very expensive, which is why stochastic gradient descent (SGD) is applied, i.e. calculating the gradient for a single sample $w^{(i)}$ as $\frac{df(w^{(i)}, \theta)}{d\theta}$. This leads to the *Bayes by Backprop* algorithm stated in algorithm 1.

Algorithm 1 Weight Update Procedure of Bayes by Backprop

- 1: Sample $\epsilon \sim N(0, 1)$
- 2: Compute $w = \mu + \log(1 + \exp(\rho)) * \epsilon$
- 3: Calculate the gradient w.r.t. μ

$$\frac{\partial f(w, \theta)}{\partial \mu} = \frac{\partial f(w, \theta)}{\partial w} + \frac{f(w, \theta)}{\partial \mu}$$

- 4: Calculate the gradient w.r.t. ρ

$$\frac{\partial f(w, \theta)}{\partial \rho} = \frac{\partial f(w, \theta)}{\partial w} \frac{\epsilon}{1 + \exp(-\rho)} + \frac{f(w, \theta)}{\partial \rho}$$

- 5: Update the networks parameters using SGD

$$\mu = \mu - \gamma \frac{f(w, \theta)}{\partial \mu}$$

$$\rho = \rho - \gamma \frac{f(w, \theta)}{\partial \rho}$$

With this algorithm it is possible to train a diagonal Bayesian neural network very similar to a deterministic NN, simply by scaling and shifting the gradient of the deterministic NN as shown in the algorithm. In the next section we look at how we can make use of the uncertainty produced by the BNN in an active learning framework.

2.2 Active Learning Theory

In conventional non-active machine learning, the learning algorithm is processing and learning from whatever training data is provided to it. This is referred to as passive learning. In that case the selection of the training data is done by a human, who is trying to select the data that is most relevant to the task that the algorithm should learn. In scenarios with very complex data or a large amount of data this can quickly become infeasible for the human, thus a common approach is to provide all available data points to the training algorithm. [18]

When we as humans are learning something about a new topic, we often follow a different approach. After we have gained some first knowledge, we gain new information interactively (e.g. a student might choose a follow up course on a topic she is interested in or a researcher might conduct an experiment that deepens her understanding). In these scenarios we actively choose the information we want to use for learning from the pool of information that is available to us (e.g. all courses offered by the university the student visits, or all experiments that are possible in the lab of the researcher). In active learning this idea is applied to machine learning in which the learning algorithm is enabled to actively choose the data points to learn from, out of a pool of available data points. In supervised machine learning this means, the algorithm chooses which data points out of a pool of unlabeled data points should be labeled and then added to the training data. This opens the possibility that the algorithm chooses those data points that are most relevant for learning the given task and omitting those data points that contain little to no additional information for solving the given task. This leads to the potential result of achieving a greater model performance with fewer training data. [19]

The advantages of this result becomes apparent when thinking about applications where data in general, but especially labeled data, is very expensive. Examples are medical applications, where patient data needs to be interpreted by a doctor, or microscopy image segmentation applications, where annotating images requires expert knowledge of microbiologists. [20]

The principle behind this idea is also referred to as pool-based active learning for supervised learning. We start by selecting a few instances of an available pool of unlabeled data $\mathcal{P}$, and ask an oracle (often a human expert) to label this small subset, which will then form the initial labeled training set $\mathcal{L}_{init}$. After training a model with a learning algorithm and $\mathcal{L}_{init}$, an acquisition function is used to decide (based on some informativeness measure) which data point from the pool should be labeled next and added to the training set $\mathcal{L}$. This approach can be continued in an iterative manner, by continuously training the model in a standard supervised way and adding new labeled points to $\mathcal{L}$. [19]

Algorithm 2 explains this in more detail.

Algorithm 2 Pool-Based Active Learning

Input:

Unlabeled pool set $P \subset \mathcal{X}$

Oracle $o : \mathcal{X} \mapsto \mathcal{Y}$ providing the true label $y \in \mathcal{Y}$ to an input $x \in \mathcal{X}$

Model $g : \mathcal{X} \mapsto \mathcal{Y}$ that will be trained on labeled data

Acquisition function $f_g : \mathcal{X} \mapsto \mathbb{R}_+$ that assigns an informativeness score to a data point x based on the model output $g(x)$.

Output:

Labeled training set $\mathcal{L}_{train}$

- 1: Randomly select a small subset S from the pool set P
 - 2: Use S and its true labels as the initial training set $\mathcal{L}_{train} \leftarrow S \cup \{o(x), x \in S\}$
 - 3: Pretrain the model g on $\mathcal{L}_{train}$
 - 4: Remove S from the pool $P \leftarrow P \setminus S$
 - 5: **while** stopping criteria not satisfied **do**
 - 6: Compute the informative scores of the pool set $\{f_g(x), x \in P\}$
 - 7: Request the label for the most informative $x^* \leftarrow \arg \max_x \{f_g(x), x \in P\}$ from the oracle
 - 8: Add x^* and its true label $o(x^*)$ to the training set $\mathcal{L}_{train}$
 - 9: Fine-tune the model g on the updated training set $\mathcal{L}_{train}$
 - 10: Remove x^* from the pool set $P \leftarrow P \setminus x^*$
 - 11: **end while**
-

The choice of the acquisition function is crucial for the success of active learning, because a bad acquisition function might not select the most informative data point that leads to the biggest increase in model performance. For active learning to be successful, sampling the data points with the acquisition function should also lead to better model performance than sampling the data points randomly from $\mathcal{P}$. The overall goal of active learning is to decrease the size of the training set while achieving the same (or better) model performance as compared to conventional non-active learning. [7]

Besides pool-based active learning there are two other scenarios where active learning can be applied. One scenario is membership query synthesis. In this scenario the learning algorithm does not select data points for labeling but generates new data points on it's own and asks the oracle for the labels. Another scenario is stream based selective sampling, where the learning algorithm only gets one data point at a time and needs to decide whether to discard it or to ask for its label and add it to the training data. [19]

Both of these scenarios do not apply well to the application of microscopy instance segmentation, because a new biological experiment typically generates a whole set of time-lapse images at once. This set can then be used as the unlabeled pool in pool-based active learning.

Batch Active Learning

An extension to this active learning procedure is to select multiple data points at once in each active learning iteration. This is called batch active learning. [19]

The benefits of batch active learning are clear when thinking about applications where the oracle is a medical specialist who is asked to label MRI scans. If we ask the specialist to label a single scan that is used to retrain the model, after which we ask the specialist again, then we can expect the overall time to create a proper model to be very long, since the availability of the specialist is also limited at some point. Further each retraining iteration is a computationally very expensive task, especially with growing network size. [19]

One approach to perform batch active learning with batch size k is to simply sample the most k informative pool data points in each iteration. In algorithm 2 this only requires a change in line 7, where we do not sample a single x^* but a whole set of size k .

$$X^* \leftarrow \arg \max_{\substack{X \subset P \\ |X|=k}} \left\{ \sum_{x \in X} f(x), X \subset P \right\}$$

In line 8 of algorithm 2 we then add the set X^* and its labels to the training set. Using this straight forward approach raises another issue about how to properly assemble a batch of size k . An acquisition function described as in algorithm 2 will likely have the property to assign similar informativeness to similar data points. Therefore, simply acquiring the k most informative data points is likely to lead to an overlap of information between the data points in the selected batch. But there are some strategies trying to overcome this issue. [19]

For example [21] suggests a novel acquisition function for Bayesian neural networks based on the acquisition function BALD (a detailed description of BALD is provided below) called BatchBALD. It computes the acquisition score for a whole set of data points at once instead for individual data points. To handle the combinatorial complexity that comes with selecting subsets, the authors make use of a greedy approximation algorithm. Using BatchBALD takes the overlap of information within the batch into account. In experiments the authors could show that BatchBALD leads to significant improvement in learning several image classification tasks by using batch active learning compared to using BALD.

Another method to find a good batch sample for active learning is the core-set approach described by [22]. In this paper, the core-set approach shows very good performance on CIFAR-10 [23] and SVHN [24] datasets with a batch sampling size of 10% of the whole dataset. On the contrary, in experiments on SVHN, reproduced by [25], the core-set approach performed worse than the random sampling baseline, which might be due to a different algorithm used for finding the core-set.

[20] offers a very straight forward solution to batch active learning, where the unlabeled pool is randomly divided into batches of preferred size. Each data-point in the batches is then scored by an uncertainty measure and the mean score for each batch is calculated. In each AL iterations the batch with the highest mean uncertainty score is selected for labeling by the oracle. The advantage of this method is, that it can be build on top of most acquisition functions. The authors showed for several chosen acquisition functions (such as entropy and uncertainty estimation based on drop-out), that the stochastic batches approach increased the dice score of a medical image segmentation task. Stochastic batches can be seen as a way to combine the computational efficiency of random sampling with the informativeness of the chosen acquisition function in a batch active learning task.

Acquisition Functions Suitable for Instance Segmentation

In the setting of supervised instance segmentation on images, the model output is typically an object index y_i for each pixel x_i in the input image x with n pixels, where the index y_i represents the object that the pixel belongs to ($y_i \in C_l$ with C_l being the set of all possible object indices)[7].

In case of a probabilistic model, a common way to determine informativeness in the acquisition function is to use the uncertainty estimation of the model. In literature there are several proposed acquisition functions for probabilistic supervised instance segmentation. [7]

1. *Maximum entropy*: Sum of the pixelwise entropies over all n pixels of the image x

$$f_{entropy}(x) = \sum_i^n \mathcal{H}(y_i|x_i)$$

with the Shannon entropy of a pixel prediction as $\mathcal{H}(y_i|x_i) = \sum_{y_i \in C_l} P(y_i|x_i) \log \frac{1}{P(y_i|x_i)}$. [7] used this acquisition function with their novel Active Bayesian Unet technique for microscopy instance segmentation and achieved a better performance compared to other acquisition functions (including BALD) for very few training images.

2. *Bayesian active learning by disagreement (BALD)*: We define the model output for a pixel y_i as the function $y_i = g_i(x, w)$, with x being the whole input image and w the model parameters. Then we can compute the BALD value of an image as the sum of pixelwise BALD values

$$f_{bald}(x) = \sum_i^n \mathcal{H}(y_i|x_i, D) - \mathbb{E}_{w \sim p(w|D)}[y_i|x, w]$$

with $\mathbb{E}_{p(w|D)}[y_i|x, w] = \int g_i(x, w)p(w|D)dw$ and the posterior of the probabilistic model $p(w|D)$. [18]

The BALD score can be seen as the mutual information between the model prediction and the model parameters. The mutual information is a measure of the dependence between those two variables. If the mutual information is high for a data point, we would gain much information about the true model parameters [21] by knowing this data point's true label. Even though [18] showed best performance with BALD on several classification datasets, [26] achieved best performance with BALD on instance segmentation tasks on only one out of four datasets. And as mentioned above in [7], using maximum entropy outperformed BALD.

3. *Sum of pixelwise standard deviations (STD)*: Supervised instance segmentation can also be done via a regression task as in [27]. In this case the probabilistic model maps each input pixel to a distribution over $\mathbb{R}$. In this case we have $y_i \in \mathbb{R}$ and a possible acquisition function is the sum of pixelwise standard deviations of the stochastic model outputs.

$$f_{std}(x) = \sum_{i=1}^n \sigma(P(y_i|x)) \quad (6)$$

[28] used this acquisition function with a network that estimates the uncertainty of an image using dropout at test time.

As mentioned before, the main goal of active learning is to reduce the amount of labeled training data. This usually also implies the goal of reducing the effort or cost for manually labeling data points. In many scenarios we can assume roughly equal labeling cost per data point. For example in a medical task, where a blood test of a patient needs to be analyzed in a laboratory, the effort stays the same for labeling each patient. But there are also other examples,

where the labeling cost varies strongly for each data point, as for example in [29]. There, the task is to semantically segment CT scans of intracranial hemorrhage, and the time it takes for annotating the objects depends linearly on the boundary length of the object areas and linearly on the number of connected components, i.e. the number connected object areas. Other examples of varying labeling cost are classifying text documents of different length [30]. A similar situation can be assumed in the annotation of microscopy images of microorganisms. Some images can contain a large colony with many thousand cells which takes a considerable amount of time to annotate, and some other images may contain only a single cell which is quick to annotate. In a similar way to the annotation of CT scans, the cell boundary length could be used as a measure for annotation effort. In case cells are of similar size and shape, also the cell count per image should correlate well with the annotation time. Still, the sampling strategies described in this section imply equal annotation cost for each data point, since the annotation effort is not incorporated in the acquisition functions. [29] solved this problem by predicting the uncertainty U and annotation cost C of each data point separately. Then batches of data points are sampled in an active learning manner, such that the sum of their uncertainties U is maximized given a budget limit for the annotation cost C . This task is also known as the 0-1 knapsack problem. It can be solved exactly in pseudo-polynomial time by an dynamic programming algorithm found by [31]. As a measure for C , [29] used a linear regression on a logarithmic transform of the boundary length and the number of connected components. The authors could show, that the cost sensitive active learning approach achieved a higher precision on the test data as by using a uniform cost distribution and the random sampling baseline.

2.3 Transfer Learning

The problem of few data samples and especially few labeled data samples in instance segmentation of microorganisms can also be tackled with a different strategy called transfer learning (TL), as described in [32]. In general, TL refers to transferring learned knowledge from one domain (source) to another domain (target). In the context of neural networks we want to transfer a models knowledge of solving a task in one domain, to a model that is able to solve a task in another domain. Transfer learning can be categorized in homogeneous TL and heterogeneous TL, where the first one describes the scenario that the same feature space is used in both domains and the latter one describes the opposite case. If TL is successful, the task on the target domain can be learned quicker by utilizing knowledge from the source domain. But we can also have negative transfer, where the opposite happens and the knowledge about one domain detains the model from learning about the other domain. For successful TL we need the source domain to be relevant for the target domain as well as the model to find the relevant information that it can reuse for the target domain. A very simple, but widely used TL method is to pretrain a neural network in one domain and then fine-tune some, or all of its parameters on the target domain. In this work, we only consider the case where the task stays the same in both domains.

Some literature work showed that by keeping the shallow layers and only fine-tuning the deep layers of the NN, the model benefits from the learned low level features (this may be curves and edges in the input images) of the pretraining dataset and can adjust the high level features to the characteristics of the new dataset in the deeper layers [33] [34]. On the contrary, [35] demonstrated that on their task of segmenting lung ultrasound images, the best fine-tuning strategy was to fine-tune all parameters of the network, though this was also the most expensive approach. The authors explain the good performance with the large differences between their pretrained dataset (non-medical images) and their target dataset (lung ultrasound images).

Some previous studies used ImageNet [36], a very large natural and man-made image dataset, as the pretraining dataset and then fine-tuned the model on medical images [37] [38]. There was also a successful attempt to pretrain a model on medical images of a particular disease of which many images are available, and then fine-tune it on only few images from another disease [39]. Therefore it seems that both, natural/man-made macroscopic images as well as images from the

same field can be a good source domain. [33] could confirm the assumption, that TL leads to better results if the source and target domain are similar. And further [34] observed that TL worked best if the target dataset was smaller.

By using a smaller learning rate in fine-tuning than during pretraining, the weights of the network change in smaller steps, allowing the network to learn the characteristics of the new domain in more detail [38]. This strategy was also applied by [34], who reduced the learning rate used for pretraining by half during fine-tuning.

2.4 Neural Network Architectures for Microscopy Image Segmentation

In the following, we introduce four neural network architectures that are commonly used for microscopy image segmentation.

U-Net [40]. A very common and often reused model architecture for microscopy image segmentation is U-Net. U-Net significantly reduced the number of annotated images needed for training while at the same time improving the segmentation precision compared to other segmentation strategies at the time of its publication. It uses a u-shaped convolutional neural network architecture, consisting of a stepwise contracting path (called encoder) and a symmetric expanding path (called decoder). In order to prevent the loss of high resolution information in the expanding path, U-Net concatenates the feature maps from the contracting path with the up-sampled outputs in each step of the expanding path. The U-Net architecture serves as the core network in all of the following architectures except for Contour Proposal Networks.

Omnipose [41]. It is a model architecture especially designed for morphology independent cell instance segmentation. The deep neural network behind Omnipose creates three model outputs for each pixel in the input image. These are first, a cell boundary probability, which is the probability of a pixel belonging to a cell boundary, second, a cell distance, which is the distance of the pixel to the closest boundary pixel of the cell it belongs to (the cell distance is zero outside of a cell), and third, a gradient flow field, which is the gradient of the cell distance and always points to the medial axis of the cell. The flow field has unit magnitude inside a cell and zero magnitude outside. Omnipose introduced a novel approach to construct the segmentation masks from those three model outputs using Euler integration. It is applicable to segment cells independently of their shape and size, because the correct construction of distance maps and flow fields is not affected by the cell morphology. Omnipose is designed and trained for cells with morphological diversity, but also varying microscopy configurations. Therefore, it not only performs well on clearly separated simple cell shapes, but also on very elongated cells and mutant cell types (e.g. *Streptomyces pristinaespiralis*). The authors point out its limitations for overlapping cells and that the cells should not be thinner than 9px and also not too large. Omnipose is also capable of 3D-segmentation. A pretrained Omnipose model is provided by the authors for usage in similar domains. [42] adapted Omnipose to efficiently fine-tune a pretrained Omnipose model for a new data domain with only one to five labeled cells from the new domain. They show that with their method only five cells are enough to reach a precision of over 0.6, whereas the precision is near zero without fine tuning.

Contour Proposal Network[3]. It was published by [3] as a framework for instance segmentation in biomedical imaging. Its architecture is briefly explained here. A backbone network takes the greyscale input image $I \in \mathbb{R}^{w \times h \times 1}$ and outputs two feature maps $P_1 \in \mathbb{R}^{s_1 \times s_1}$ and $P_2 \in \mathbb{R}^{s_2 \times s_2}$, with P_1 having higher resolution ($s_1 > s_2$). Then a classification head takes P_2 as input and outputs a classification map $O \in \mathbb{R}^{s_2 \times s_2}$ that determines whether a pixel in P_2 is representing an object or not. Further a contour regression head takes P_2 as input and outputs a contour representation map $C \in \mathbb{R}^{s_2 \times s_2 \times (4N+2)}$ defining a contour for each pixel in P_2 . A contour representation $c \in \mathbb{R}^{4N+2}$ has degree N and is defined in [43]. The 2D coordinates of

the contour can be calculated by a sine and cosine transformation function with c defining the parameters of this function. The higher the degree N of a contour is, the more detailed this contour can approximate the outline of a real cell, but also the larger the contour representation c gets. After this step all contour representations from C are extracted from those pixels and the classification map O identifies an object. The result is a list of proposed contour representations, which are transformed back into pixel coordinates. A second regression head takes P_1 as input and creates a refinement tensor $V \in \mathbb{R}^{s_1 \times s_1 \times 2}$. This tensor is used to refine the contour pixel coordinates. In this way, it is possible to obtain more detailed contours, even with lower contour degree N . Because of the way the proposed contours are extracted, multiple contours might represent the same object. Therefore, as a final step redundant contours have to be removed by a non-maximum suppression (NMS) algorithm.

According to [3] the strength of CPNs lie in the detection of cell boundaries even if they are partially invisible. It can also handle touching objects, with no clear gaps or boundaries between them, very well. Another advantage is that CPNs, in contrast to Omnipose, also enable the detection of overlapping cells, because the proposed contours are predicted independently of each other, thus a pixel can belong to two or more contours. [3] evaluated the cross-data generalization of their model by training different models on one dataset and applied them to another dataset. In these experiments they compare the generalization of a CPN to a standard U-Net and a Mask R-CNN (another efficient instance segmentation method [44]). It could be shown that the CPN achieves overall a better generalization, measured by the average F1-score.

In [45] the team further extended CPNs with an additional uncertainty estimating head, that estimates four localization parameters for each proposed boundary of an object. This uncertainty estimation, together with the detection score from O is used in the NMS algorithm to remove redundant contours more effectively. This modification has only a slight affect on the models performance on the validation set (measured by the F_1 score), but the uncertainty prediction adds valuable information that can be used for other applications. One of this potential applications is efficient training data selection in an active learning framework. The uncertainty estimation could serves as an input for the acquisition function.

CNN based distance predictions [27]. This work builds upon the idea of [46], who formulated the problem of instance segmentation not as a classification but as a regression task. A neural network, in this case a U-Net, learns to predict a cell distance for each input pixel. The cell distance is defined as the normalized distance of the pixel to its closest background pixel. This leads to a representation of the input picture where the cell distance is high in the center of a cell and goes towards zero as the pixels approach the cell boundary (see Figure 2, c.).

The next step is to compute the actual image segmentation from this cell distance representation. For this, a post-processing algorithm is added that makes use of the *watershed transformation*, that was first applied on segmentation problems by [47]. A mathematical description of this algorithm is given in subsection 4.1. A good estimation of the cell distance representation should sustain two assumptions. First, the cell centroids should be located at local maxima of the representation and second, two cells should be separated by a significant drop (the shape of a valley) in the cell distance. In the first post-processing step the algorithm uses these two assumptions to decide which local maxima are cell centroids (also called seeds). This decision is controlled by the parameter th_{seed} . In the second step the post-processing algorithm applies the watershed transformation to find the cell regions for each seed. This is controlled by the parameter th_{cell} . Both parameters can have values between 0 and 1 and can cause bad segmentation results if chosen wrongly as shown qualitatively in Table 1.

[27] extended this method by adding a second parallel decoding head. The first head still aims to predict the cell distance representation. The second head aims to predict a neighbor distance representation. The neighbor distance is the inverse normalized distance of a cell-pixel to the nearest pixel of the neighboring cell. So the neighbor distance has high values where two cells are very close to each other and low values for cell areas which are far away from other cells.

	Seed Threshold	Cell Threshold
too low parameter value	incorrect fragmentation of the actual cells	background areas will falsely be recognized as cells
too high parameter value	incorrect combination of multiple cells	cell areas will falsely be recognized as background

Table 1: The resulting behavior of the cell instance segmentation if the post-processing parameters th_{seed} and th_{cell} are chosen wrongly. The parameter values lie between 0 and 1.

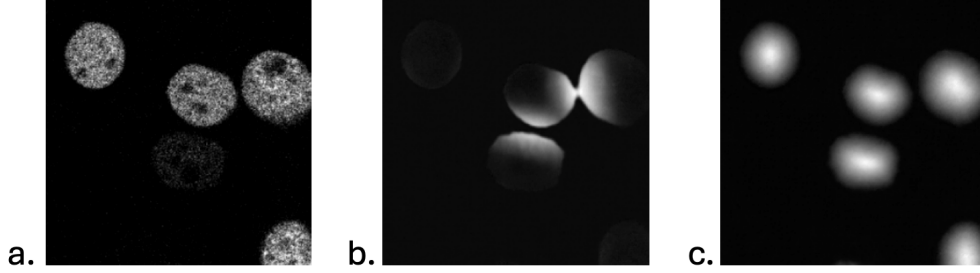


Figure 2: Microscopy image (a.) of cells with its neighbor distance representation (b.) and cell distance representation (c.). Images are taken from [27].

The background has the value zero in an exact neighbor distance representation (see Figure 2, b.). Both the cell distance and the neighbor distance representation are additionally smoothed by a Gaussian kernel with standard deviation σ . The goal of this extension is to increase the models ability to separately identify touching cells. For this, the squared neighbor distances are subtracted from the cell distances for identifying the cell centroids (first post-processing step). The second step of post-processing is computed in the same way as in [46] with cell distances only.

Usually a labeled dataset contains the ground truth segmentation masks, so one needs to be able to transform them into the ground truth cell and neighbor distances in order to train the model in a supervised manner. The ground truth cell distances can be easily computed from the segmentation masks by applying the Euclidean distance transform [48]. [27] also provides a number of different transformations, including the Euclidean distance transform to compute the ground truth neighbor distance shown in figure Figure 3. In step (b) the cell itself and the background is converted to foreground while the other cells become background. Then, in (c) and (d), the distance transformation is applied and cropped to the area of the current cell. After inverting and normalizing in (e), the cell transformations are aggregated (f) and very close cells are connected via greyscale closing (g) [49]. As a last step in (h), the resulting distances are taken to the power of three to ensure a steeper decline within cells.

All the above described methods and architectures are based on neural networks. Therefore it is possible to apply the Bayesian framework *Bayes by Backprop* [10] to any of those architectures to arrive at a Bayesian neural network. The obtained uncertainty estimation can then be used in an active learning framework.

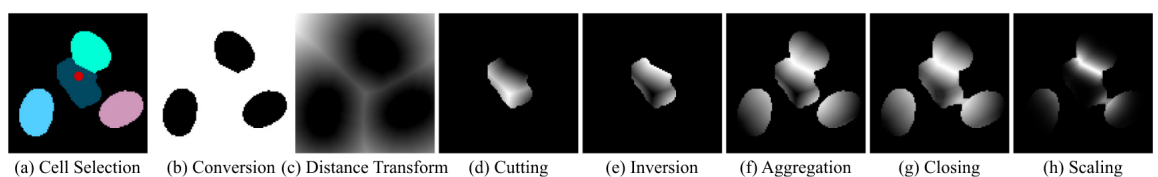


Figure 3: Transformations to compute the neighbor distance. Image taken from [27].

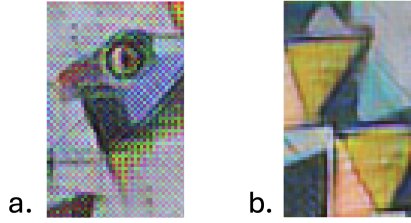


Figure 4: Image with (a.) and without (b.) checkerboard artifacts. Image taken from [50].

2.5 Checkerboard Artifacts

[50] discovered, that convolutional neural networks sometimes produce unwanted grid-like patterns in the output images. An example is shown in Figure 4. The authors identified two processes where these checkerboard artifacts are created. Firstly, those patterns can be created during a forward pass through one or multiple transposed convolutional layers by an uneven map of pixels from the input feature map to the pixels of the output feature map. This uneven mapping of pixels is created when the kernel size is not divisible by the stride of the layer. But even with a divisible kernel size, checkerboards can be produced by the learned weights of the kernels. The second process where these patterns can be created is during backpropagation of convolutional layers. It can be shown, that those two processes are in some cases mathematically equivalent [51]. Another type of checkerboard can be caused by max pooling layers, which are typically of higher frequency [50]. For solving the issue [50] proposed to either use a transpose convolutional kernel size that is divisible by the stride, or to replace the transpose convolutional layer and instead resize the input feature maps by e.g. nearest neighbor interpolation followed by a normal convolutional layer. In some cases it was also observed, that the model was able to learn to suppress the artifacts by carefully adjusting the weights in the kernel. Other proposed solutions to avoid checkerboard artifacts involve other variants that replace the transposed convolutional layer [51] [52]. Since the U-Net architecture involves upsampling layers, there is a chance of observing the artifacts if no measures are taken against it.

3 Related Work

In this section we take a closer look at some related literature and proposed methods.

3.1 Related Active Learning Literature

[7] uses an approach similar to this work to increase labeled training data efficiency for microscopy image segmentation tasks. This is done by a pool based active learning framework where the informative estimation for sampling new images is gained through uncertainty based acquisition functions. Their main contribution is to model uncertainty through a Bernoulli BNN (utilizing Monte Carlo dropout from [6]) and batch normalization with stochastic parameters [53]. Using a U-Net architecture, dropout is placed after each max-pool layer and batch normalization is placed after two consecutive convolutional layers. This architecture was empirically found to perform best. The network directly outputs the object class for each pixel without any post-processing step. Model prediction is performed with Markov Chain Monte Carlo (MCMC) approximation, where n forward passes from different samples of the stochastic model parameters are averaged.

[7] compares four acquisition functions namely, maximum entropy, BALD, committee posterior KL-divergence and committee posterior Jensen divergence. The first two are summarized in section 2 and are based on entropy. The latter two are based on divergence between the standard prediction $P(y|x, D)$ and the MCMC prediction $\mathbb{E}_{P(w|D)}[y|x, \theta]$, using the KL-Divergence and the Jensen divergence between these two terms respectively. Further the results are shown for four different microscopy image datasets and the Dice coefficient [54] is used for evaluation.

The active learning experiments have the following schema. Two images are taken as the initial labeled training set. In each active learning iteration the two most informative (decided by the respective acquisition function) images are added to the training set. The network parameters are refined in each iteration for about 60 epochs (early stopping criterion depends on the dice score of the validation set). The learning rate is set to 0.001 and Adam optimizer is used.

The results show that active learning with the mentioned acquisition functions was only able to significantly beat iterative training using random sampling on two out of four datasets. In the two experiments where the active acquisition lead to an improvement the divergence based acquisition functions, especially the committee posterior Jensen divergence performed best. Entropy and BALD outperformed random sampling only in one of the four experiments significantly. Finally the proposed AB-UNet with the four acquisition functions was compared to two other methods, namely cost-effective active learning [28] and sample selection and conditional generative adversarial networks [55]. Here the usage of entropy and committee posterior Jensen divergence as acquisition functions outperformed those methods.

It is also worthy to note that [7] observed that the estimated uncertainty increased with the number of object classes - an indication that the uncertainty and thus informativeness increases with increasing number of objects (cells of microorganisms) on an image.

3.2 Related Instance Segmentation Literature

This thesis uses a variation of the network architecture of [27], which is why the results of this method are discussed here. The methodology for improving microscopy instance segmentation by adding a second regression head to a U-Net has been discussed in section 2.4. The authors participated in the IEEE ISBI 2020 Cell Tracking Challenge [56] with their work. The network was trained on a mixed training set consisting of samples from different datasets provided by the challenge. The performance of the model was evaluated on each dataset separately. The Cell Tracking Challenge includes the Fluo-N2DL-HeLa dataset, a set that is also evaluated in this thesis (see section 5.1). The segmentation performance is measured by the Jaccard similarity index (SEG) with an IOU threshold of 0.5 (see section 4.3) on a pixel level and by the normalized acyclic oriented graph matching measure for detection (DET) [57] on an object level. As explained earlier, the parameters for the watershed post-processing can have a strong influence on the SEG. [27] points out that the parameters can be optimized for each individual dataset but to preserve comparability the parameters were set to $th_{seed} = 0.5$, $th_{cell} = 0.09$ and the standard deviation of the Gaussian kernel $\sigma = (1.5, 1.5)$. During training, different data augmentation methods are applied namely flipping, scaling, rotation, contrast changing, blurring and adding noise. The training images are normalized by min-max transformation in the range $[-1, 1]$. The proposed method achieved good results in the IEEE ISBI 2020 Cell Tracking Challenge ranking in the top three highest SEG scores for 7 out of 19 datasets. The results for some of the datasets on a held out test set are shown in Table 2. The example of Fluo-N3DH-CE, where only few images were used in the training set, suggests a good generalization ability of this method. One stand out property resulting from adding the neighbor distance predictions is that the model is also able to learn from information about the closeness of cells without the need of many touching cells in the training data. Thus, even though there were only few examples of touching cells in the training data, the model was still able to separate touching cells in the test data well.

Another method, where it is worth to look at the results in this context in more detail is Omnipose [41]. The goal of this method is to segment microscopy images of diverse morphologies. The architecture and method has been summarized in section 2. For performance measurement the IOU (here called JI for Jaccard index, see section 4.3 for a detailed description) was used with varying thresholds. The authors could demonstrate in Figure 5, that segmentation with Omnipose performs very well independent of the cell area, which corresponds approximately to

	SEG	DET
BF-C2DL-HSC	0.750	0.974
Fluo-N3DH-CE*	0.729	0.930
Fluo-N2DL-HeLa	0.895	0.992
BF-C2DL-MuSC	0.702	0.977

Table 2: Results of CNN based distance predictions on different datasets on a pixel level (SEG) and on an object level (DET). Fluo-N2DL-HeLa (HELA) is also used in this thesis. For Fluo-N3DH-CE less than five images of this dataset are used in the training set.

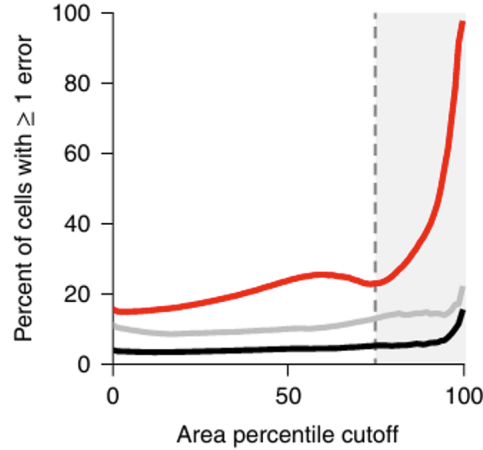


Figure 5: Segmentation performance over cell size for Omnipose (black) compared with its predecessor method "Cellpose" (red) [58] and a combination of both (grey). Measured by percent of cells that have at least one segmentation error over each cell area percentile. Graph is taken from [41].

cell length on their dataset. This underlines the strength of Omnipose for segmenting elongated cells. Omnipose was also evaluated on the highly diverse Cellpose dataset [58]. The results are shown in Figure 6. Even though the performance of Omnipose on Cellpose data is weaker than on the more homogeneous Worm_omni, it still reaches an IOU of approximately 0.63 with an IOU threshold of 0.5. It is also noteworthy, that the authors offer to use their models, which are pretrained on various datasets. They suggest, that it is sufficient to use the pretrained models without the need of fine-tuning in scenarios with similar target data. The results of Omnipose on the Cellpose dataset are especially relevant, because the same dataset (CPOSE) is used to evaluate the Bayesian active learning approach in section 6. It has to be noted though, that the exact images used in this thesis may vary, since the dataset is regularly extended with new images. Nevertheless, the property of high diversity including non-microscopy images and multi-channel fluorescence images remains the same.

4 Methodology

This work uses a NN with U-Net architecture to predict cell distance maps from the input images. Further we apply the Bayesian learning algorithm from [10] to the U-Net resulting in a BNN. We use the uncertainty estimation of the Monte Carlo (MC) sampled cell distance maps from the BNN in the acquisition function of a pool based active learning framework to fine-tune the BNN. As acquisition function, the *sum of pixelwise standard deviations* (STD) is used. The strategy of sampling the training images for active learning varies in each experiment and is explained in more detail in section 4.2. A post-processing algorithm computes the segmentation masks from

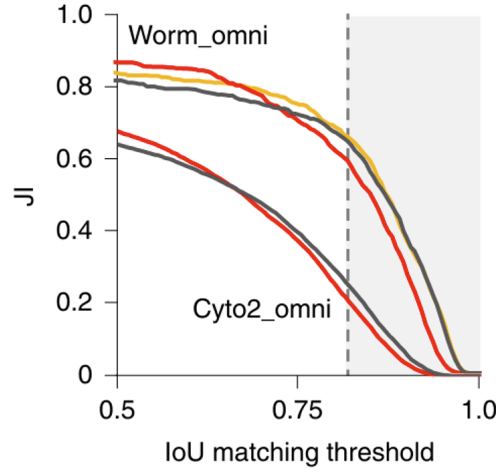


Figure 6: Jaccard Index (JI/IOU) as a function of IOU threshold for the Cellpose dataset (Cyto2_omni) and a *C. elegans* dataset (Worm_omni) containing worm-shaped nematodes. Omnipose (grey) is compared with the Cellpose model (red) [58] and an Omnipose model trained on Worm_omni and bacterial data (yellow). The important result here is Omnipose on Cyto2_omni data, because we can compare it with the results of this thesis on the same dataset. Graph is taken from [41].

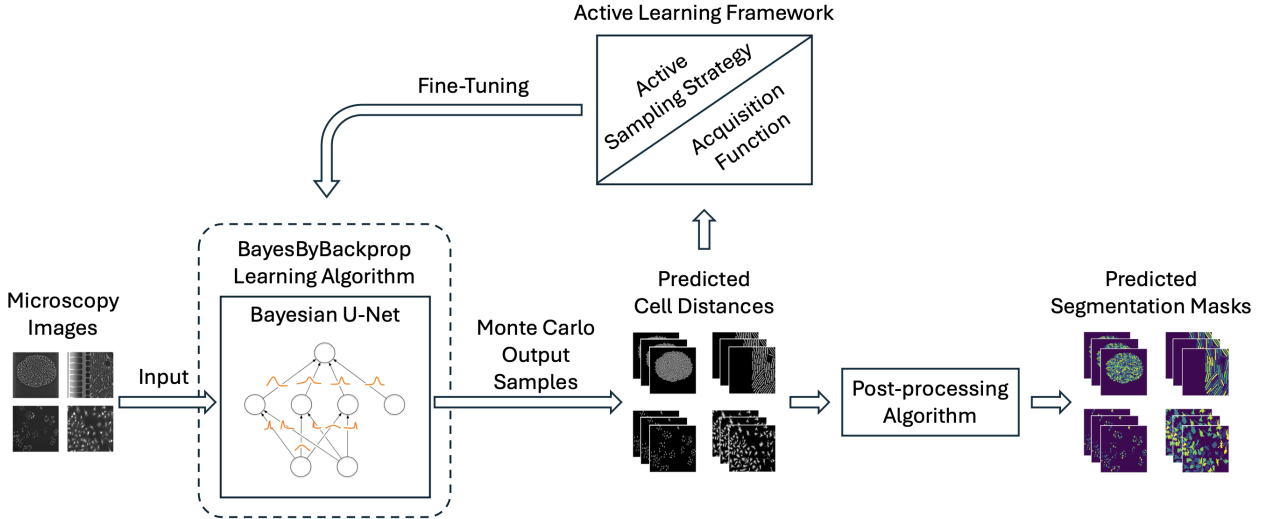


Figure 7: High-level overview of the training pipeline.

the cell distance maps. Figure 7 gives a high-level overview of the training pipeline.

The segmentation performance of the BNN that was fine-tuned through active learning is then compared to the segmentation performance of a BNN that is fine-tuned with randomly sampled training images. Through this approach, we aim to find an answer to the posed research question.

4.1 Model Architecture and Post-Processing Algorithm

A conceptual overview of the used architecture and post-processing step is shown in Figure 8. For datasets with a high abundance of touching cells, the proposed architecture from [27] is a good choice, because the second head, that predicts the neighbor distance, is improving the NN’s ability to detect touching cells as separate objects. However, the aim of this work is to study the applicability of active learning to gain higher segmentation accuracy and not improving the network architecture. Thus, for the sake of simplicity, the neighbor distance head was only used for the experiments in subsection 6.2 and omitted in all further experiments. Omitting the neighbor distance maps leads to shorter runtimes and easier interpretation of loss and uncertainty

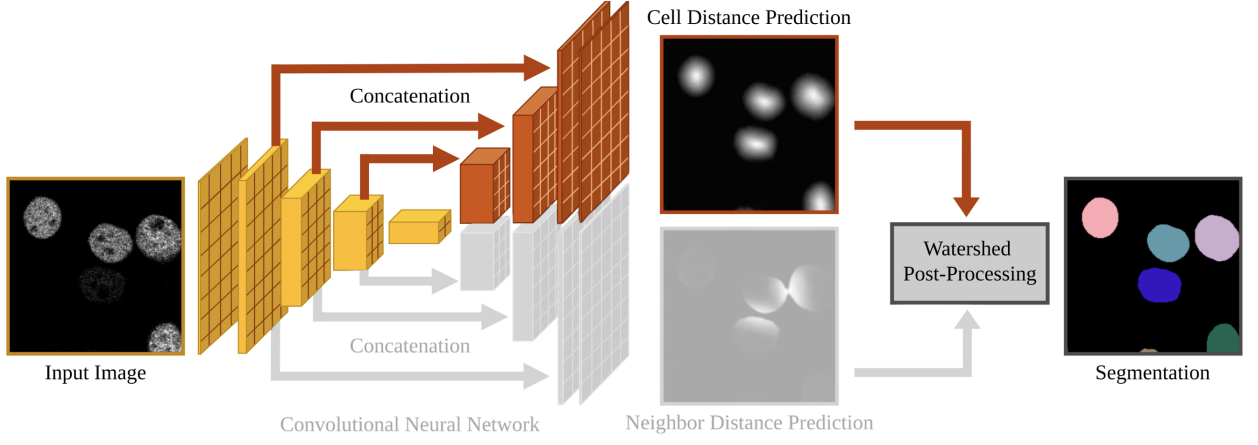


Figure 8: Conceptual overview of the architecture and post-processing step. The second prediction head (grey) was added by [27], but is only used in section 6.2. All other experiments use the cell distance maps alone. The image is modified and originally taken from [27].

due to the absence of a second loss term from the neighbor distance. The ground truth cell and neighbor distance maps, that are needed for supervised training of the BNNs, are calculated from the ground truth segmentation masks according to [27] summarized in section 2.4. The structure of the used U-Net architecture is shown in Table 3. We quantify the influence of the changes that we made to the architecture proposed by [27] in section 6.2.

The U-Net architecture is used to construct a BNN with the learning algorithm from [10]. We chose a standard normal distribution $N(0, 1)$ over the network weights for the prior. The posterior distribution is a diagonal Gaussian distribution $N(\mu, \text{diag}(\sigma)^2)$ with randomly initialized mean μ and initial standard deviation $\sigma = 0.001$ for each dimension. The loss function, used in the data dependent term of the VFE, is the pixelwise L1-loss. For a single input image the L1-loss can be expressed as $L_{x,L1}(w) = \sum_i^n \|y_i^* - y_i\|_1$ with y_i^* being the true label of pixel i and y_i being the label that is predicted by the NN for pixel i with the parameters w and input image x containing n pixels.

For the results in section 6.2, we implemented the post-processing algorithm to include the neighbor distance. This is shown in algorithm 3. By taking $\tan P_{neighbor}^2$ in step 2 of algorithm 3, we steepen the values of $P_{neighbor}$ as they approach 1. The transformation further sets very small values of $\tan P_{neighbor}^2$ to 0 and keeps the values smaller or equal than 1. For all other experiments algorithm 4 is used, which does not make use of neighbor distance maps.

4.2 Active Learning Framework

Transfer Learning Scenario

In a laboratory, a biologist usually conducts a new in vitro experiment, where the output of such an experiment is a new dataset of unlabeled images. We now wish to make use of datasets that have already been labeled and verified by experts in the past, to train a model that can segment the new dataset. For this, we use a model that is pretrained on old datasets and fine-tuned with only few images of the new dataset. This transfer learning scenario is simulated in this work by first pretraining models on each chosen dataset using the whole training set and then fine-tuning each of the models on one of the other datasets in a pool based active learning manner. To decide, which two datasets are combined for pretraining and fine-tuning, we compared all possible combinations in section 6.3. During fine-tuning all neural network layers are trained, and no layers are frozen. As discussed in section 2.3, literature in the field of medical instance segmentation suggests that this is a valid approach [35]. We do not focus on optimizing TL, this is rather an extension to make our experiments more useful for the application in a lab.

Layer	Activation (ReLu)	No. of Filters	Kernel (3x3)	Stride	Padding (zero padding)
1-Conv2D	-	64	-	1	-
2-Conv2D	-	64	-	1	-
3-MaxPool	-	64	-	2	-
4-Conv2D	-	128	-	1	-
5-Conv2D	-	128	-	1	-
6-MaxPool	-	128	-	2	-
7-Conv2D	-	256	-	1	-
8-Conv2D	-	256	-	1	-
9-MaxPool	-	256	-	2	-
10-Conv2D	-	512	-	1	-
11-Conv2D	-	512	-	1	-
12-MaxPool	-	512	-	2	-
13-Conv2D	-	1024	-	1	-
14-Conv2D	-	1024	-	1	-
15-Transposed Conv2D	no activation	512	2x2	2	no padding
16-Conv2D	-	512	-	1	-
17-Conv2D	-	512	-	1	-
18-Transposed Conv2D	no activation	256	2x2	2	no padding
19-Conv2D	-	256	-	1	-
20-Conv2D	-	256	-	1	-
21-Transposed Conv2D	no activation	128	2x2	2	no padding
22-Conv2D	-	128	-	1	-
23-Conv2D	-	128	-	1	-
24-Transposed Conv2D	no activation	64	2x2	2	no padding
25-Conv2D	-	64	-	1	-
26-Conv2D	-	64	-	1	-
27-Conv2D	no activation	1	1x1	1	no padding

Table 3: Detailed overview of the used neural network architecture. The cell entry '-' means, that the default setting is applied which is shown in parentheses in the corresponding column title. Zero padding is always applied with size one (the feature maps get padded on all four sides by one element having the value zero). Each layer is followed by batch normalization [53], except the last layer. The input feature maps of the convolutional layers in the upsampling blocks are concatenated by the output feature maps of the corresponding downsampling blocks, as proposed by [40].

Algorithm 3 Post-Processing Algorithm with Neighbor Distance

Input:

Cell distance map P_{cell}
Neighbor distance map $P_{neighbor}$
Seed threshold th_{seed}
Cell threshold th_{cell}

Output:

Cell Segmentation Mask

- 1: Smooth the cell distance map with a Gaussian filter with $\sigma = 0.5$

$$\hat{P}_{cell} = P_{cell} \otimes G(\sigma)$$

- 2: Transform $P_{neighbor}$ by the following pixelwise operation.

$$\hat{P}_{neighbor} = \begin{cases} 0 & \text{if } \tan P_{neighbor}^2 < 0.05 \\ 1 & \text{if } \tan P_{neighbor}^2 > 1 \\ \tan P_{neighbor}^2 & \text{otherwise} \end{cases}$$

- 3: Obtain the seed candidates map P_{seeds} by applying the seed threshold to each pixel of $\hat{P}_{cell} - \hat{P}_{neighbor}$ as follows.

$$P_{seeds} = \begin{cases} 1 & \text{if } \hat{P}_{cell} - \hat{P}_{neighbor} > th_{seed} \\ 0 & \text{otherwise} \end{cases}$$

Connected areas where P_{seeds} is 1 are considered as one seed (cell centroid).

- 4: Remove seeds which are smaller than 10% of the average seed area.
- 5: Compute the region to flood P_{mask} by applying the cell threshold to each pixel of $\hat{P}_{cell}$ as follows.

$$P_{flood} = \begin{cases} 1 & \text{if } \hat{P}_{cell} > th_{cell} \\ 0 & \text{otherwise.} \end{cases}$$

- 6: Use watershed according to [47] to compute the cell segmentations from P_{flood} and P_{seeds} .
-

Algorithm 4 Post-Processing Algorithm without Neighbor Distance

Input:Cell distance map P_{cell} Seed threshold th_{seed} Cell threshold th_{cell} **Output:**

Cell Segmentation Mask

- 1: Smooth the cell distance map with a Gaussian filter with $\sigma = 0.5$

$$\hat{P}_{cell} = P_{cell} \otimes G(\sigma)$$

- 2: Obtain the seed candidates map P_{seeds} by applying the seed threshold to each pixel of $\hat{P}_{cell} - \hat{P}_{neighbor}$ as follows.

$$P_{seeds} = \begin{cases} 1 & \text{if } \hat{P}_{cell} > th_{seed} \\ 0 & \text{otherwise} \end{cases}$$

Connected areas where P_{seeds} is 1 are considered as one seed (cell centroid).

- 3: Remove seeds which are smaller than 10% of the average seed area.
- 4: Compute the region to flood P_{mask} by applying the cell threshold to each pixel of $\hat{P}_{cell}$ as follows.

$$P_{flood} = \begin{cases} 1 & \text{if } \hat{P}_{cell} > th_{cell} \\ 0 & \text{otherwise.} \end{cases}$$

- 5: Use watershed according to [47] to compute the cell segmentations from P_{flood} and P_{seeds} .
-

Acquisition Function and Oracle Simulation

In our experiments, we want to sample training images from the fine-tuning dataset by active learning. To simulate the oracle that annotates unlabeled images when asked, part of the fully labeled dataset was held back as the "unlabeled" pool set and the correct segmentation masks were revealed when requested by the acquisition function. As acquisition function we use the *sum of pixelwise standard deviations* (STD) as stated in section 2.2. By this, we can compute the acquisition score directly from a set of MC outputs for each pixel. If we use *maximum entropy* or *BALD* we would need to interpret the cell distance as the probability for a pixel belonging to a cell. This would be an additional assumption that is not required when using STD. For computing the STD we first generate m cell distance maps

$$y_j = f(x, w_j), \text{ with } w_j \sim q(w|\theta) \text{ and } j = 1, 2, \dots, m$$

where $f(x, w_j)$ is the Bayesian neural network with the weight vector w_j sampled from the approximate posterior by Monte Carlo sampling. Then the pixelwise standard deviation can be calculated from the set of m generated outputs $(y_j)_{j=1}^m$. In all experiments m is set to 10. In our acquisition function we do not incorporate the varying annotation costs of the images and implicitly assume an uniform annotation cost distribution. This is obviously a wrong assumption when looking at the TOIAM example images in Figure 9. Nevertheless, as a first step, we decided to evaluate the results without considering varying annotation costs.

Active Learning Algorithm

As described in section 4.2, we want to simulate the realistic situation in a laboratory by a transfer learning scenario, where a model gets pretrained on the whole training set of one dataset and

then fine-tuned on another dataset. We aim to keep the number of labeled images used for fine-tuning low by active learning. We created a training set for each dataset used in this work, that contains 60% of the images of the respective dataset (see section 5.1 for more details). During active learning we use the training set of the fine-tuning dataset and assume it to be the unlabeled pool set. The AL procedure for fine-tuning is described in algorithm 5.

In one AL iteration, MC predictions are made on the pool set, then the acquisition function samples new images, the simulated oracle labels the images and finally the updated training set is used to fine-tune the original pretrained model. This means in each AL iteration the weight updates of previous fine-tuning iterations are discarded and the training starts over from the initially pretrained model. This is done in order to avoid getting stuck in a local minima.

Active Learning Experiment Design

In our main experiments we compare four different models for each fine-tuning dataset. The models are called "*Single Iteration AL*", "*Multi Iteration AL*", "*Uniform Sampling with pretrained Model*" and "*Uniform Sampling without pretrained Model*". "*Single Iteration AL*" and "*Multi Iteration AL*" are initialized with the weights of a model that was trained with the pretraining dataset. Both models are then fine-tuned with AL using the STD acquisition function. "*Single Iteration AL*" is fine-tuned with one AL iteration. "*Multi Iteration AL*" is fine-tuned with four AL iterations, but we only evaluate the performance after the last AL iteration. The size of the fine-tuning training set is equal in both cases. The models "*Uniform Sampling with pretrained Model*" and "*Uniform Sampling without pretrained Model*" are fine-tuned with a training set that is sampled uniformly at random (UNI). The model "*Uniform Sampling with pretrained Model*" was initialized with the weights of a pretrained model, whereas "*Uniform Sampling without pretrained Model*" was randomly initialized (without pretraining). Table 4 gives an overview of the four models. Additionally we compare those four models to a "*reference model*" that was trained on the whole training set of the fine-tuning dataset.

Further, the experiments are conducted for two different fine-tuning training set sizes. The training set sizes are chosen differently for each dataset and can be taken from Table 5.

In some of the experiments we sample more than one training image in each AL iteration. We are aware, that this batch sampling approach poses the risk of overlapping information between the sampled images, as elaborated in section 2.2. We leave it to future studies to minimize this risk by applying some of the discussed measures in section 2.2. However, by varying the size of the batches in *Single Iteration AL* and *Multi Iteration AL*, we may get a sense of the effect of batch sampling.

Note that by comparing "*Uniform Sampling with pretrained Model*" with "*Single Iteration AL*" and "*Multi Iteration AL*", we can investigate whether using the STD acquisition function is beneficial over uniform sampling (UNI). This is what we use to answer the posed research question. By comparing "*Uniform Sampling without pretrained Model*" with "*Uniform Sampling with pretrained Model*" we want to investigate whether using a pretrained model has a benefit over training a new model from scratch, at least in the uniform sampling case.

The learning rate was reduced during fine-tuning compared to pretraining. As recommended by [38], this allows more detailed weight updates. The hyperparameters regarding the architecture and learning algorithm used during fine-tuning can be taken from Table 6. The hyperparameters used for the pretrained models can be taken from Table 7. In all training runs the number of weight updates was chosen in a way that the loss curves (training and validation loss) stayed constant for at least 1000 weight updates. By observing the difference between training and validation loss, we noticed that overfitting did not increase or decrease if we continued with training for much more than 1000 weight updates. For convenience, this allows us to set the number of weight updates constant for each dataset, without being at risk that overfitting increases during training. Via grid-search over kl_{weight} , we observed that for the choice of $kl_{weight} < 10^{-5}$, the variance over the cell distance maps almost vanished and thus, the predicted uncertainties of the BNN

Algorithm 5 Used Active Learning Procedure

Input:

Pretrained BNN g_{pre} , where $g_{pre}(x, w^{(j)})$ denotes one segmentation mask from input image x and MC sampled weights $w^{(j)}$.

Unlabeled pool image set $\mathcal{X}_{pool}$

Number of MC samples m , AL batch size k and AL iterations T

STD acquisition function from Equation 6 f_{std}

Training algorithm $\text{train}(g, \mathcal{L}_{train})$ to train model g with training set $\mathcal{L}_{train}$

Output:

Actively sampled training set $\mathcal{L}_{train}$

BNN $g_{current}$ fine-tuned on $\mathcal{L}_{train}$

1: Set the pretrained model to be the current model: $g_{current} \leftarrow g_{pre}$

2: Define empty training set $\mathcal{L}_{train} \leftarrow \{\}$

3: **for** $t = 1$ to T **do**

4: Predict m segmentation masks for each image x in $\mathcal{X}_{pool}$

$$\mathcal{Y}_{pool,x} \leftarrow \{g_{current}(x, w^{(j)}) \mid j = 1, \dots, m\}, \text{ with } x \in \mathcal{X}_{pool}$$

5: Compute the acquisition scores $\mathcal{Z}_{pool}$ from the MC predictions of each $\mathcal{Y}_{pool,x}$

$$\mathcal{Z}_{pool} \leftarrow \{f_{std}(\mathcal{Y}_{pool,x}) \mid x \in \mathcal{X}_{pool}\}$$

6: Select the k pool images $\mathcal{X}_{sampled}$, which have the highest scores in $\mathcal{Z}_{pool}$

$$\begin{aligned} \mathcal{X}_{sampled} \leftarrow \{x \in \mathcal{X}_{pool} \mid f_{std}(x) \geq \min(\mathcal{Z}_{top\ k}), \text{ where } \mathcal{Z}_{top\ k} \subseteq \mathcal{Z}_{pool} \text{ and } |\mathcal{Z}_{top\ k}| = k, \\ \text{s.t. for all } z_1 \in \mathcal{Z}_{top\ k} \text{ and all } z_2 \in \mathcal{Z}_{pool} \setminus \mathcal{Z}_{top\ k}, z_1 \geq z_2\}. \end{aligned}$$

7: Get the true segmentation masks $\mathcal{Y}_{sampled}$ from the oracle for the sampled images $\mathcal{X}_{sampled}$

8: Update the training set

$$\mathcal{L}_{train} \leftarrow \mathcal{L}_{train} \cup \{(x_i, y_i) \mid x_i \in \mathcal{X}_{sampled}, y_i \in \mathcal{Y}_{sampled}, i = 1, 2, \dots, k\}$$

9: Update the pool set $\mathcal{X}_{pool} \leftarrow \mathcal{X}_{pool} \setminus \mathcal{X}_{sampled}$

10: Fine-tune g_{pre} with the updated training set and assign as $g_{current}$

$$g_{current} \leftarrow \text{train}(g_{pre}, \mathcal{L}_{train})$$

11: **end for**

	Single Iteration AL	Multi Iteration AL	Random Sampling with Pretraining	Random Sampling without Pretraining
Number of AL Iterations	1	4	1	1
Acquisition Function	STD	STD	UNI	UNI
Pretraining Model used	yes	yes	yes	no

Table 4: Overview of active learning settings during fine-tuning. We conducted four different experiments with two different training set sizes per dataset. STD is the sum of pixelwise standard deviation, UNI is uniform sampling at random.

Fine-Tuning Set Size	TOIAM	MSEG	HELA	CPOSE
Small Training Set Size	1	4	1	16
Large Training Set Size	4	8	4	96

Table 5: Number of fine-tuning training images used for each dataset. In case of Multi Iteration AL, the training set increases in each iteration. The number given in this table is the final training set size in the last AL iteration.

became very small and difficult to measure. But the choice for kl_{weight} had only little influence on the behavior of the complexity cost and the error cost during training. To prevent a very low uncertainty, the kl_{weight} was further manually set to a fixed value of 10^{-3} . Which dataset was chosen for pretraining and which for fine-tuning was decided in subsection 6.3 and summarized in Figure 17.

4.3 Evaluation Metrics

To measure the performance of a model, the Intersection over Union (IOU) measure and the F1 score with an IOU-threshold of 0.5 are used. The IOU is calculated as suggested by [59].

$$IOU = \frac{1}{k} \sum_{i=1}^k \frac{|G_i \cap S|}{|G_i \cup S|}$$

With the condition of choosing S such that $|G_i \cap S| \geq 0.5 \cdot |G_i|$.

Here, $|G_i|$ is the pixel area of the i^{th} object (cell) of the ground truth segmentation and S is any object of the predicted segmentation, that fulfills the above condition. Note that for each ground truth object G_i only one predicted object S can fulfill the condition. The term in the summation is the Jaccard index. The IOU of an image is calculated as the average IOU of the

Weight Updates in each AL Iteration	4000
Learning Rate	10^{-4}
Batch Size	*Equal to training set size
KL-weight	10^{-3}

Table 6: Chosen hyperparameters during fine-tuning of the pretrained models. The learning rate was reduced compared to pretraining (where it was $5 \cdot 10^{-4}$).

*The batch size was chosen equal to the size of the fine-tuning training set, except for the experiment with CPOSE with 96 fine-tuning images, where the batch size was set to 12.

Training Dataset	Learning Rate	Batch Size	Weight Updates	Epochs	KL-Weight
TOIAM	$5 \cdot 10^{-4}$	20	$2 \cdot 10^4$	138	10^{-3}
MSEG	$5 \cdot 10^{-4}$	10	$2 \cdot 10^4$	10000	10^{-3}
HELA	$5 \cdot 10^{-4}$	20	$2 \cdot 10^4$	2858	10^{-3}
CPOSE	$5 \cdot 10^{-4}$	20	$2 \cdot 10^4$	870	10^{-3}

Table 7: Hyperparameters used for pretraining models on the four datasets. MSEG requires a smaller batch size because the training set has less than 20 images. Learning rate and batch size were found via grid search.

individual cells. For measuring the segmentation accuracy of a whole dataset with many images, we average the IOU measure of the individual cells and not of the images. In this way, images are weighted proportionally to their number of cells. This is done, since we want to know how well we can segment all objects in the dataset and not how well the segmentation performs on one image on average. We also use the IOU on the individual ground truth objects G_i to define correctly identified objects as true positives (TP), predicted objects S with no matching ground truth object as false positives (FP) and overseen ground truth objects as false negatives (FN). From this we calculate the F1 score as

$$F1 = \frac{2TP}{2TP + FP + FN}$$

Because the F1 score only takes into account if an object is identified or not and ignores how accurate the segmentations are on a pixel level, we expect the F1 score to be higher than the average IOU. Further, the F1 score is able to penalize false positives, whereas the IOU is not influenced if an additional object is predicted in the background of the image. This might play a role, where images have artifacts in the background which can easily be mistaken as cells (see the MSEG image with index 2 in Figure 9).

5 Experimental Setup

5.1 Used Data

The experiments of this work are conducted with four different microscopy image datasets. The dataset are **Tracking-one-in-a-million** (short **TOIAM**) from [60], **microbeSEG** (short **MSEG**) from [61], **Fluo-N2DL-HeLa** (short **HELA**) generated by [62] and accessed via [56] and **Cell-pose** (short **CPOSE**) generated by [58] and accessed via [63]. Figure 9 displays some sample images of the datasets.

TOIAM consists of five chronologically ordered time-lapse sequences, that were generated during the growth of *C. glutamicum* colonies with a time resolution of one minute. HELA was generated by creating two time-lapse sequences of HeLa cells. The images of MSEG are image snippets of two separate growing colonies without chronological order. CPOSE is a large collection of images ranging from proteins to a few non-microscopy images of fruits which also vary in size, resolution and imaging technique. Some images of CPOSE origin from the same colony, but in general the big variety of images characterize this dataset. Table 8 gives an overview about some of the properties of the datasets. Note that TOIAM uses the same microscopy technique as MSEG. Moreover CPOSE includes at least some epifluorescence images like HELA.

The images of all datasets are cleaned such that the images have the same size, value range and number of channels. For the U-Net architecture it is convenient if the input image edge length is a power of two 2 with the exponent p larger than the number of max-pool layers in

Dataset Name	Occurring Microbes	Microscopy Technique	Time Resolution	Spacial Resolution	Image Size
TOIAM	<i>C. glutamicum</i>	phase contrast	1 min	$0.072 \frac{\mu m}{px}$	938x1094
MSEG	<i>B. subtilis</i> , <i>E. coli</i>	phase contrast	15/20 min	$0.07 \frac{\mu m}{px}$	320x320
HELA	HeLa	epifluorescence	30 min	$0.645 \frac{\mu m}{px}$	1100x700
CPOSE	diverse	epifluorescence, bright field, other microscopy, non-microscopy	diverse	diverse	diverse

Table 8: Properties of the used datasets.

the networks encoder. Otherwise padding needs to be applied to concatenate the feature maps in the decoder. During cleaning, the microscopy images and their segmentation masks were first resized to 512x512 pixels by the function `transforms.Resize()` from the `torchvision` package. Nearest-neighbor interpolation was used for up-scaling smaller images. Next, the microscopy images were normalized to the range $[-1, 1]$ using min-max normalization. Some of the microscopy images from CPOSE have two channels. Those images were additionally merged to one channel by averaging the pixels of each channel. Further, the cell indices of all segmentation masks were renumbered such that the indices follow a consecutive numbering from 1 to number of cells n_{cells} while background pixels have the value zero.

The datasets were each split into a training, validation and a test set of the size 60%, 20% and 20% respectively. Because of the exponential distribution of cells in each sequence of TOIAM and HELA, in some cases, randomly splitting the set did not generate representative subsets. For example, the test set may contain no image with many cells. In order to prevent this, these two datasets are split as follows: First the chronologically ordered image sequences are split into buckets of five subsequent images. Then each of these 5-image-buckets are split uniformly at random into a training (three images), validation (one image) and a test set (one image). The final training, validation and test sets were then created by unifying the sets of all the buckets. Following this approach, the cell count distribution of the original dataset is guaranteed to be maintained in each subset. The drawback of this splitting method is, that the images in the sets are not independently distributed, because the selection of one image depends on the images that were selected before. It thus violates the i.i.d. (independent and identically distributed) assumption. The images from CPOSE and MSEG were split uniformly at random, since there is no exponential cell count distribution.

Characteristics of the Used Datasets

The four chosen microscopy image datasets possess very different properties. Generally, the properties are more homogeneous within TOIAM and HELA, and more heterogeneous within MSEG and CPOSE. From Figure 9 we see that TOIAM and HELA possess less background structures, but both datasets exhibit a growing colony over time. Especially TOIAM has large variations in the number of cells per image. MSEG on the other hand contains many background structures from the chip and also other artifacts. CPOSE has a large variety of cells of different morphology and size, and also varying background patterns. CPOSE also contains cells with a visible nucleus that creates a pattern inside the cell.

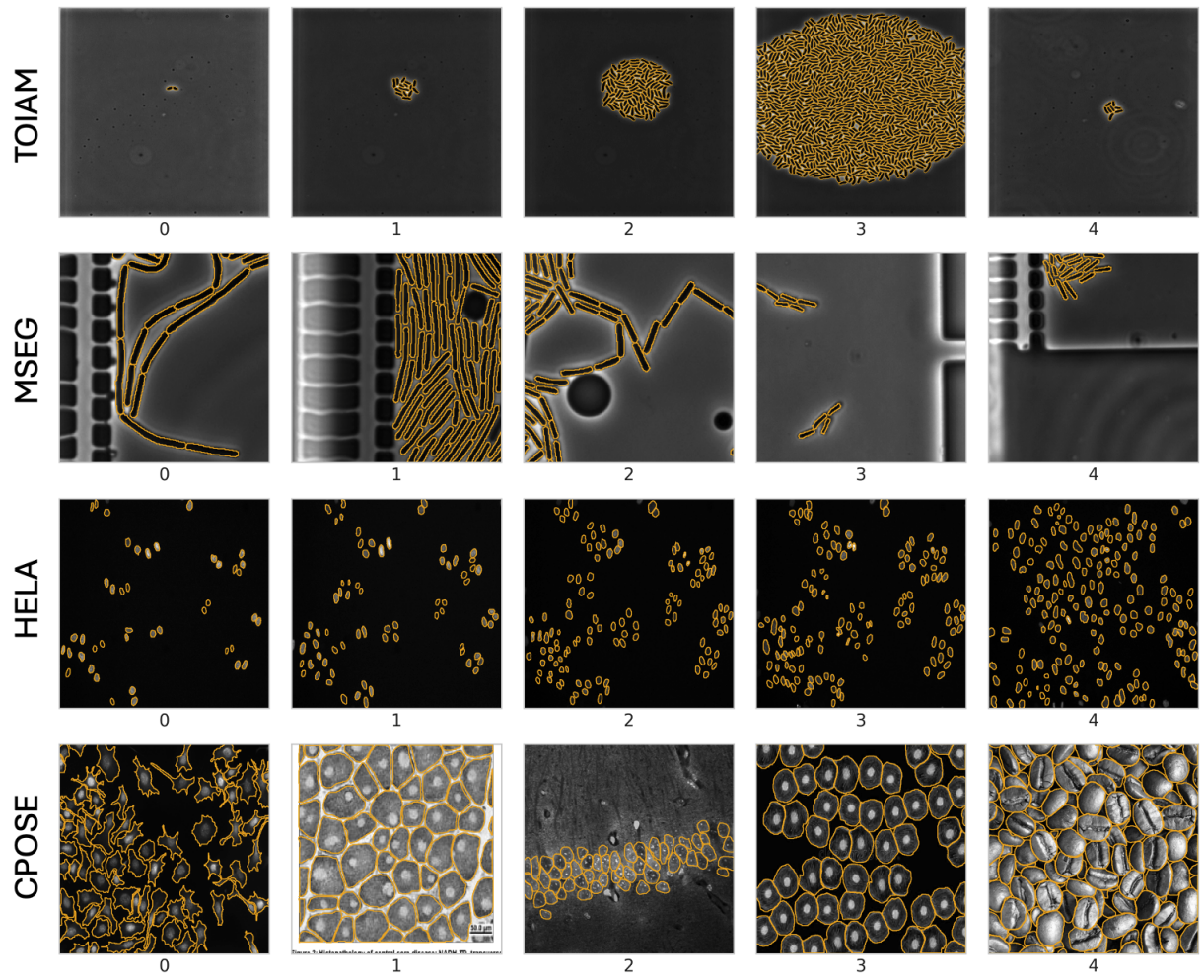


Figure 9: Example images for each dataset. The orange lines indicate the ground truth segmentation.

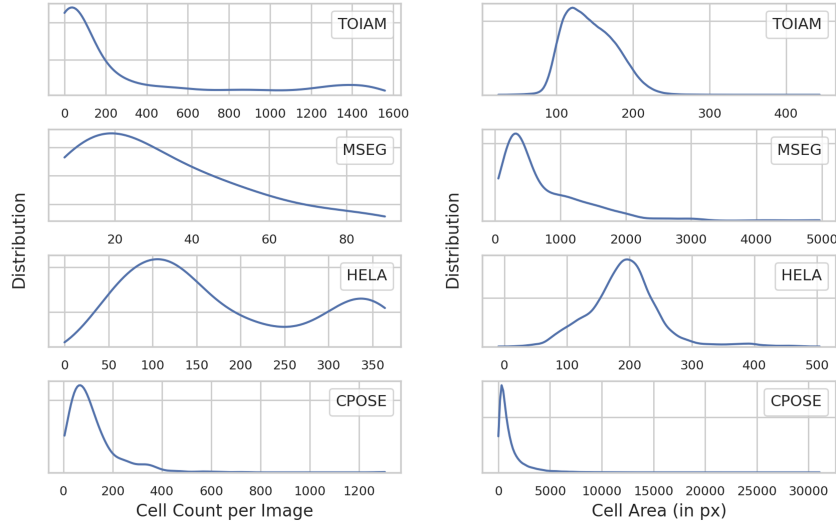


Figure 10: Distribution of cell count per image (left) and cell area measured in pixels (right). The distributions are generated by the `kdeplot()` function from the `seaborn` package with the argument `bw_method='scott'`. It is a Gaussian kernel density estimation, where the variance is computed using Scott's rule [65].

We quantify the dataset characteristics by four statistical measures. The density distribution of the number of cells per image quantify the abundance of cells in each dataset. The density distribution of the cell area in pixels quantify the cell size in each of the datasets. Further the histograms over the background pixel values serve as a measure for how complex and abundant background structures are in each dataset. The last measure is the histogram over the edge proportion of the cells. The edge proportion of a cell is calculated as

$$\frac{C}{\sqrt{\pi \cdot A}},$$

where the cell circumference C is set in relation to the cell area A measured in pixels. The edge proportion is a modification of the Polsby-Popper score [64]. For a perfect circle the edge proportion is 2 and for any cell that assumes more complex shapes (elongated, star-shaped, etc.) this value increases (given that the cell is large enough to avoid effects from pixel-discretisation).

The four statistical measures are shown in Figure 10 and Figure 11. We see that TOIAM has, due to its exponential growth many images with smaller number of cells, but also images with a very high cell count. MSEG and HELA have a comparably low cell count, but a more even distribution over all images. The cell area shows a much smaller mean and variance in TOIAM and HELA as compared to MSEG and CPOSE (note the different scales on the x-axis). MSEG and CPOSE have more variations in cell size. For the background pixel distribution, both TOIAM and HELA have a narrow peak at low pixel values, which indicates a uniformly colored background with little structures or artifacts. Especially MSEG, but also CPOSE have a wider background pixel distribution explaining the different background patterns (see images of MSEG in Figure 9) and shades of gray (see CPOSE image 1 and 2 in Figure 9). According to the edge proportions, HELA contains the least complex cell shapes. TOIAM, due to the elongated cells, has higher edge proportion values. Most cells in CPOSE also do not vary strongly from circular shapes, but there are cells that have edge proportions greater than three. MSEG has the highest edge proportion values, simply due to its very long stick-shaped cells. Table 9 summarizes the characteristics of the four datasets qualitatively.

5.2 Development Environment

The experiments were conducted on two different computing machines and on one workstation, which was used for development and small tasks such as creating plots. Single experiments, that

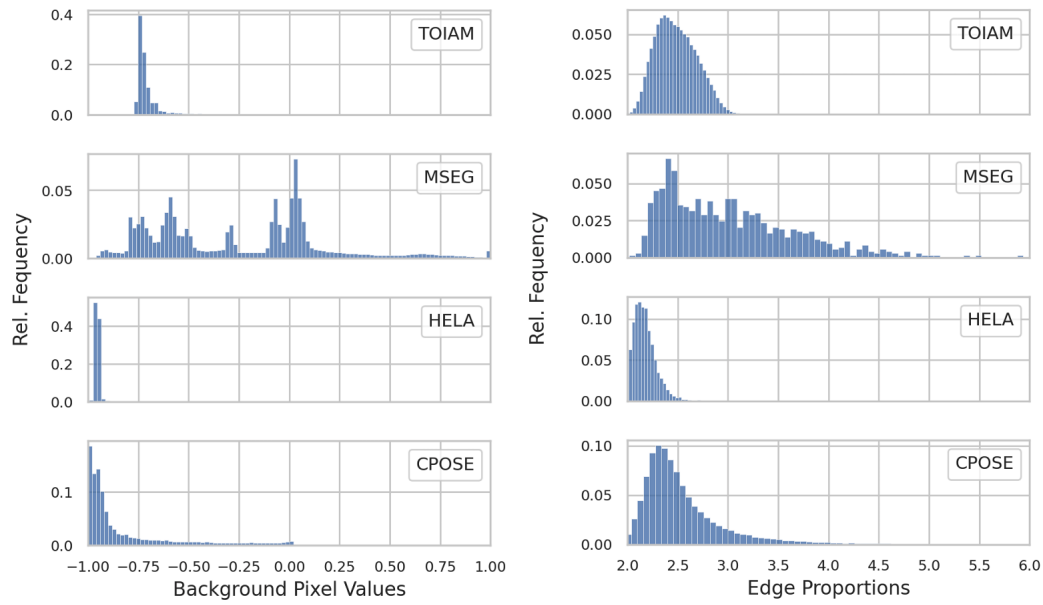


Figure 11: Histograms of the pixel values of background pixels (left) and edge proportions (right).

	# Images	Cell Abundance	Cell Size	Cell Shape	Background
TOIAM	4000	many cells exponential distr.	small and homogeneous	simple and homogeneous	homogeneous background
MSEG	22	very few cells	heterogeneous	simple and homogeneous	heterogeneous patterns
HELA	184	few cells bimodal distr.	small and homogeneous	simple and homogeneous	homogeneous background
CPOSE	608	many cells very diverse	very heterogeneous	complex and heterogeneous	heterogeneous patterns and color

Table 9: Qualitative properties of the datasets.

Library	Version	Usage
PyTorch (torch)	1.12.0	Model generation, data transformation, loss function
Torchvision	0.17.2	Image transformation during cleaning
MLflow	2.10.2	Tracking experiments (metrics, parameters and models)
Pandas	2.2.2	Aggregation of results
Seaborn	0.13.2	Plotting
Matplotlib	3.7.1	Plotting
Pillow	10.2.0	Image preprocessing
Tifffile	2023.4.12	Image preprocessing

Table 10: Overview of the most important used libraries and their versions.

required more computing power ran on a GPU-server, and extensive parallel experiments which required large computing resources such as grid search or the final active learning experiments were carried out on a supercomputer. The supercomputer runs with the operating system Rocky Linux 8 and each of the 192 computing nodes of the supercomputer have

- 2 CPUS: AMD EPYC 7742, 2x 64 cores, 2.25 GHz
- Memory: 512 (16x 32) GB DDR4, 3200 MHz
- 4 GPUS: 4x NVIDIA A100 GPU, 4x 40 GB HBM2e
- Network interface: 2x InfiniBand HDR (NVIDIA Mellanox Connect-X6)

The GPU server runs with Ubuntu 22.04.4 LTS and in total consists of

- 256 CPU: AMD EPYC 7742, 2x 64 cores, 2.25 GHz
- Memory: 1 TB
- 6 GPUS: NVIDIA A100 GPU, 80 GB HBM2e
- OS: Ubuntu 22.04.4 LTS

All software components, including the model and the learning algorithm, the active learning framework, the evaluation of the experiments and plotting the results were programmed in Python version 3.10.12. Table 10 gives an overview about the most important libraries and versions that were used. Further JupyterLab (version 4.0.1) was used as a sandbox environment to run small experiments. For bigger experiments, the CUDA toolkit in combination with the Slurm batch system was used to run the program on distributed GPUs in parallel.

6 Results

The results of this work are split into six sections. The main results, our active learning experiments, can be found in section 6.5. The following list gives an overview about this chapter.

- Section 6.1. In this section we show how we select the hyperparameters of the post-processing algorithm. We compare two methods.
- Section 6.2. In this section we investigate the effect of our changes to the NN architecture from [27] on the segmentation performance.
- Section 6.3. In this section we decide which dataset is used for pretraining and which dataset is used for fine-tuning in our active learning experiments.

- Section 6.4. In this section we show, that for two of our four datasets, the BNNs performance depends on the number of cells that are present in the training set.
- Section 6.5. These are the main results of this work. We compare our active learning method to uniform sampling on four different fine-tuning datasets.
- Section 6.6. Some of our BNNs produce unwanted checkerboard artifacts. In this section we show a method how to prevent this.

6.1 Post-Processing Hyperparameter Selection

In our main experiments, we want to measure the performance of different BNNs using the IOU. But the computation of the IOU requires to first apply the post-processing algorithm to the predicted cell distance maps of the BNNs to further create the segmentation masks. We then use the IOU to measure the similarity between the predicted and the ground truth segmentation masks. As mentioned in section 2.4, the post-processing algorithm requires to set two thresholds th_{seed} and th_{cell} . From [27] we know, that if these thresholds are chosen wrongly, the algorithm can create faulty segmentation masks, even if the given cell distances are correct. From this, the question arises by how much the post-processing algorithm can reduce the IOU, even if the cell distance maps are predicted correctly. And further, how to best choose the thresholds th_{seed} and th_{cell} . These questions are answered in this subsection.

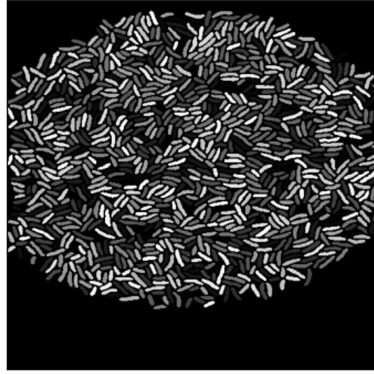
We compare two methods to find the values for th_{seed} and th_{cell} . One is Otsu’s method for image thresholding [66] where we use the same Otsu value for both thresholds. The other method is to maximize the IOU via grid-search over the two thresholds. In both methods, we pass the ground truth cell distances through the post-processing algorithm to compute the segmentation masks. We then compute the IOU between these generated segmentation masks and the ground truth segmentation masks. For optimal thresholds, we expect an IOU of 1, since the masks should be identical. Figure 13 shows the comparison in IOU of the two methods. Due to the much higher IOU, we chose grid-search to calculate the post-processing thresholds.

Upon investigating why the results are so different, we found that the resulting masks from using Otsu’s method show, that pixels at the cell borders are falsely classified as background, making the predicted cells thinner than the ground truth cell masks (see Figure 12). This indicates that the Otsu-threshold is too high for th_{cell} .

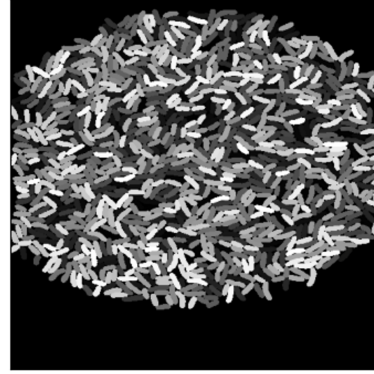
Different datasets may require different thresholds as suggested by [27]. Therefore, we optimize the post-processing thresholds via grid-search for each dataset separately. For this, the cell distance maps were generated by BNNs trained on the full training set of the corresponding dataset. The results are averaged over four training runs (over four different models). The acquired thresholds, shown in Table 11, are used in all further experiments.

Dataset	Seed Threshold th_{seed}	Cell Threshold th_{cell}
TOIAM	0.6	0.1
MSEG	0.4	0.05
HELA	0.6	0.1
CPOSE	0.5	0.05

Table 11: Optimized hyperparameters for watershed post-processing for each dataset. Hyperparameters were found via grid search.



(a) Otsu's method
(1073 segmented cells)



(b) Ground truth masks
(1072 segmented cells)

Figure 12: Segmentation mask generated by applying the post-processing algorithm with Otsu thresholds onto the ground truth cell distances (a) compared to the ground truth segmentation mask (b). Image taken from TOIAM dataset.

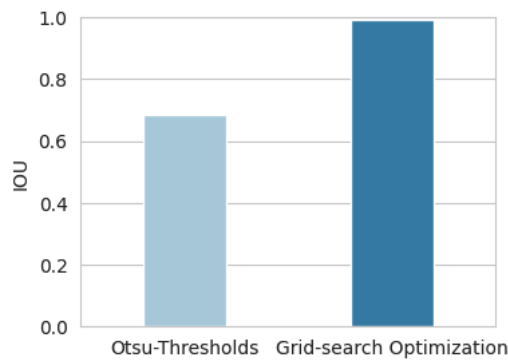


Figure 13: Selecting the post-processing thresholds th_{seed} and th_{cell} with Otsu's method vs. via grid search. The IOU is computed over all images of the first sequence of TOIAM. Both values were calculated as the average over five runs.

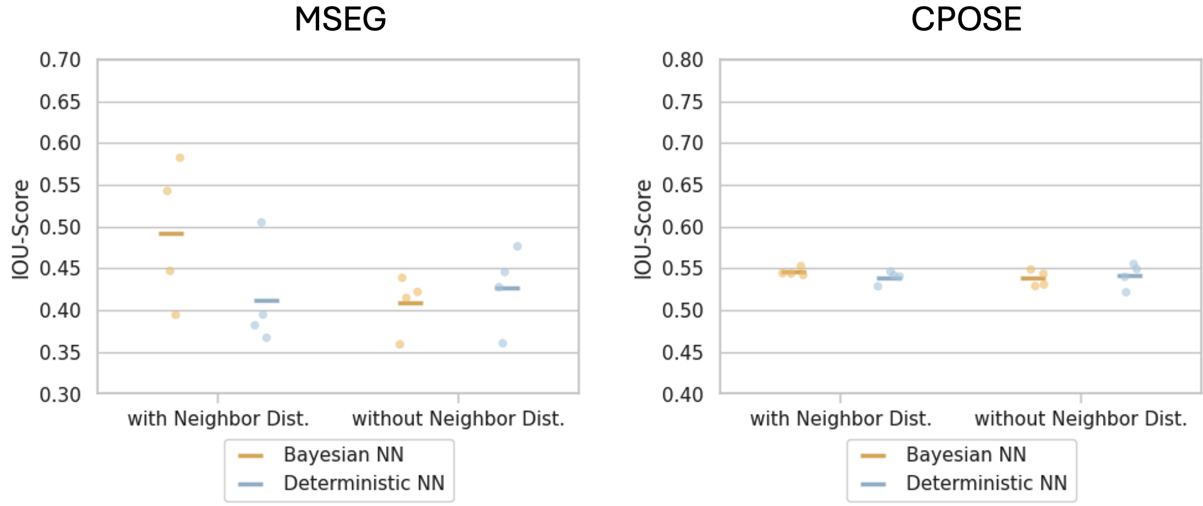


Figure 14: Comparison of the segmentation performance of different network architectures resulting from the two architectural changes with/without neighbor distance and Bayesian/deterministic neural network.

6.2 Neural Network Architecture Comparison

The architecture we use is based on [27]. It is therefore helpful to understand the influence of the architectural changes (described in section 4.1) that we made on the performance of the neural network. One change is omitting the neighbor distance. Other changes are made in the course of applying the Bayesian inference learning method from [10]. Both of these changes can be present or absent in the neural network independently of each other, which gives us four possible architectural combinations:

- deterministic NN without neighbor distance
- deterministic NN with neighbor distance
- Bayesian NN without neighbor distance
- Bayesian NN with neighbor distance

We compare the IOU for networks trained on the CPOSE and MSEG dataset. This analysis was not conducted for the datasets HELA and TOIAM, because the segmentation accuracy is always very high and differences in the IOU can hardly be noticed. Figure 14 shows the NN performance on the test set of CPOSE and MSEG for NNs of each of the four architecture combinations.

For CPOSE adding the neighbor distance has only a slight positive effect, even though there are many touching objects in the dataset. The effect of using a Bayesian NN over a deterministic NN also seems to have only a small influence. For MSEG the neighbor distance improves the performance clearly for the case where a Bayesian NN is used, but also the spread is highest. In the following experiments, every neural network is a Bayesian neural network without neighbor distance, unless stated differently.

6.3 Pretraining Dataset Selection

The active learning experiments in section 6.5 are conducted using a dataset for pretraining the BNN (we call this the pretraining dataset) and a dataset for fine-tuning the BNN (we call this the fine-tuning dataset). The motivation for this transfer learning scenario is described in section 4.2. We want to fine-tune BNNs using our AL method on each of the four chosen datasets. This means, for each of the four fine-tuning datasets, we need to select a pretraining dataset.

The purpose of pretraining a model is to improve the models performance on the fine-tuning dataset. This is why we want to select the pretraining dataset out of the remaining three datasets, such that it increases the BNN's performance on the fine-tuning dataset by the most.

We can measure the performance of a pretrained BNN on the fine-tuning dataset before or after the BNN was fine-tuned with the fine-tuning dataset. We call this the pre-fine-tuning performance and the post-fine-tuning performance respectively. Because we are interested in improving the performance after fine-tuning, we select that dataset for pretraining, which increases the post-fine-tuning performance by the most. Additionally, we also look at the pre-fine-tuning performance, because it is interesting to see whether the pretraining dataset that increases the pre-fine-tuning performance by the most, is the same as the the pretraining dataset that increases the post-fine-tuning performance by the most.

To determine this, we pretrained four BNNs on each dataset using the whole training set. Table 7 shows the hyperparameter used for pretraining the models. The performance of those BNNs were then evaluated on all datasets, which gives us the pre-fine-tuning performance in Figure 15. For example the cell in the second row and first column of Figure 15 shows the F1 or IOU of a model that was pretrained on CPOSE and evaluated on the HELA dataset. Next, we fine-tuned each of the four BNNs on the three remaining datasets and evaluated the performance on the respective fine-tuning dataset. This gives us the post-fine-tuning performance that is shown in Figure 16.

The number of training images used for fine-tuning was 2 (HELA), 2 (TOIAM), 32 (CPOSE) and 8 (MSEG). The values were averaged over 10 fine-tuning runs. Fine-tuning the models with just a small amount of images significantly increased the cross-dataset performance of all models. Note that the datasets HELA and TOIAM are predicted well after fine-tuning with only two images, regardless of which pretraining model was used. Also note that the segmentation performs very poorly on CPOSE with any model. This is most likely due to the large variety of images present in the dataset.

On the contrary the cross-dataset performance without fine-tuning on the other datasets is very low with one exception. The model pretrained on CPOSE achieves a very high F1 score (0.91) on HELA without any fine-tuning. The reason to this might be that CPOSE contains some epifluorescence microscopy images as does HELA. In addition, CPOSE is a very large and homogeneous dataset which may generalize better.

By comparing the pre-fine-tuning performance in Figure 15 and the post-fine-tuning performance in Figure 16, we see that for only two out of four fine-tuning datasets, the best pretraining dataset stays the same in both figures. This tells us, that simply using the pre-fine-tuning performance to determine which dataset suits best for pretraining is not recommended.

Based on the results in Figure 16, we chose the combinations of pretraining and fine-tuning datasets, that we use for the active learning experiments. The combinations are chosen with respect to the best performing combinations with one exception such that MSEG is used as pretraining dataset for TOIAM. This combination is almost as good as using HELA for pretraining, but it is interesting to see the performance of our smallest dataset for pretraining in our active learning experiments.

6.4 Dependency of the Segmentation Performance on the Cell Count

Before we analyze the results of our active learning experiments in section 6.5, we want to look at the dependency of the segmentation performance of a BNN, on the number of cells present in the training set.

From section 5.1, we know that the images in TOIAM and HELA have very little background structures. This may mean that the background areas of an image are less informative for training a BNN and we may assume, that cell areas and areas close to cells are more informative. If this assumption holds, it means that the information content per image, that can be used to improve the BNN during training, is higher the more cells an image contains.

In this section we show that this assumption holds for the datasets TOIAM and HELA.

To validate the assumption of informativeness correlating with cell count for TOIAM and HELA we trained several BNNs (without pretraining) with different number of cells in the training

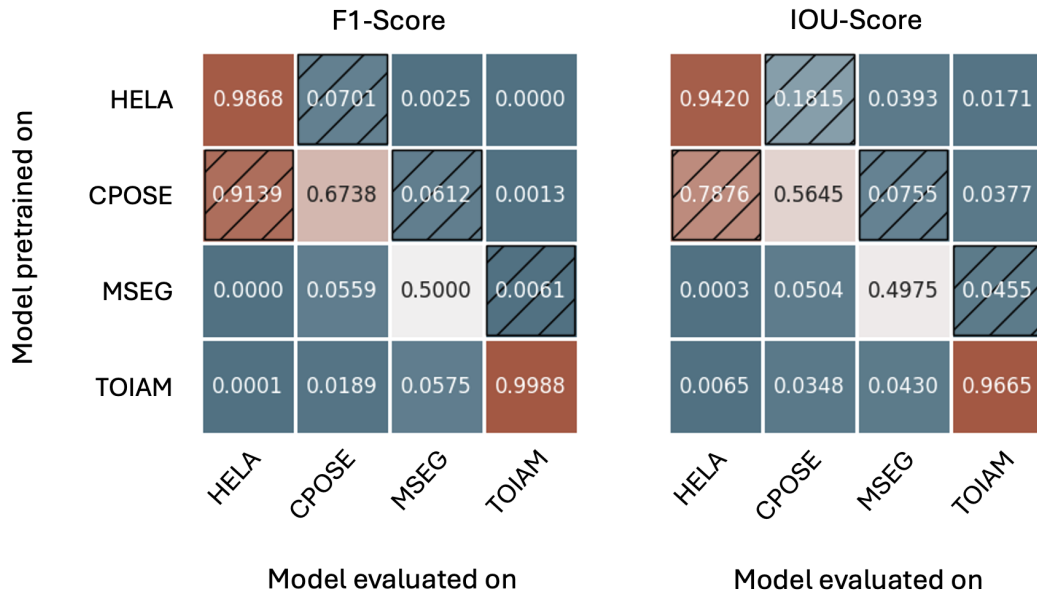


Figure 15: Segmentation performance of four models pretrained on each dataset and evaluated on all other datasets. The striped cells are the highest off diagonal scores in each column, which marks the best pretrained model for a given fine-tuning dataset. The off-diagonal cells are always evaluated on the whole dataset (training, validation and test set) whereas the diagonal elements are evaluated on the test set only.

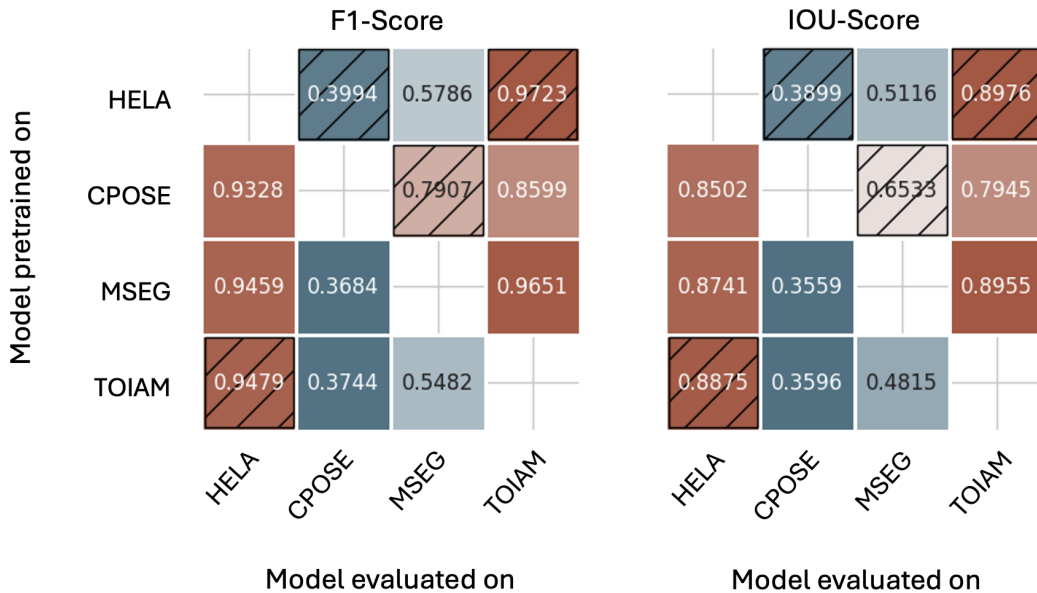


Figure 16: Segmentation performance of four models pretrained on each dataset and fine-tuned on all other datasets. The models are evaluated on the test set of the respective fine-tuning dataset. The striped cells are the highest off-diagonal scores in each column. Compare with Figure 15.

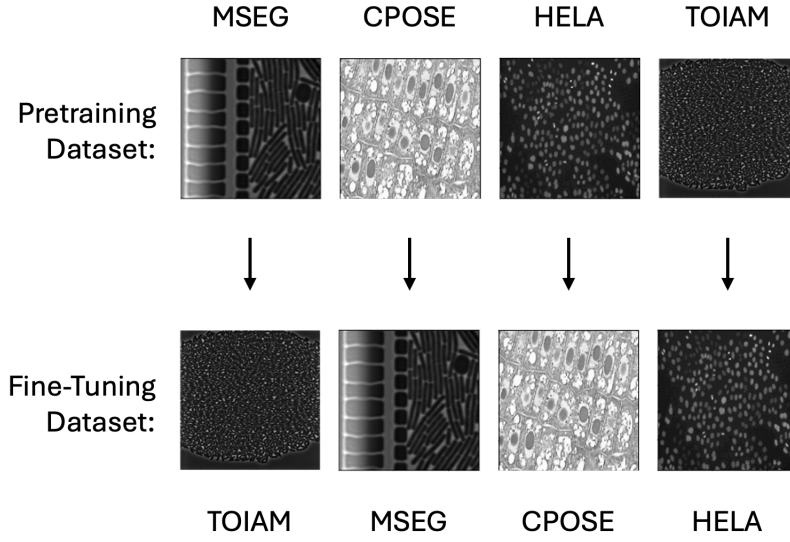


Figure 17: Four models are trained on a pretraining dataset and fine-tuned on a fine-tuning dataset. This figure shows the dataset combinations used in the active learning experiments in section 6.5.

set and compared their performance on the test set of the respective dataset. The number of images used for training was set to 2 in all training runs.

The result of this experiment for HELA is shown in Figure 18. We see that the more cells are present in the training set, the higher is the segmentation performance on the test set images. Note, that using a very high amount of cells (726 cells) does not result in a large performance gain compared to using a moderate amount of cells (266 cells). However, the performance drops rapidly if the training set contains only a low amount of cells (86 cells).

The result for TOIAM is shown in Figure 19. By looking at the average F1-score, we also see better performance of the BNN, the more cells are present in the training set. As in the experiment with HELA, the performance decreases the most if only few cells (4 cells) are present in the training set. Also note, that the BNN trained on images with many cells (average of 1552 cells per training image), achieves a high performance on all test images, irrespective of their cell count.

From these experiments we know, that the informativeness of an image correlates with the number of cell present in that image. Thus, in case of TOIAM and HELA, a working active learning algorithm may sample training images that contain a high number of cells.

Another outcome of this experiment is, that in case of TOIAM and HELA, a very small training set size of only two images is sufficient to train a randomly initialized BNN to achieve a good test performance. Even two training images that contain a moderate amount of cells are enough to achieve an average F1-score above 0.8.

6.5 Active Learning Results

This subsection shows the main results of this thesis. We compare two active learning approaches ("*Single Iteration AL*" and "*Multi Iteration AL*") and two uniform sampling approaches ("*Uniform Sampling with pretrained Model*" and "*Uniform Sampling without pretrained Model*"). The details of these approaches are explained in section 4.2. Each model is pretrained on one dataset and fine-tuned on another dataset. The results are shown in Figure 20. The name above each chart refers to the dataset that was used for fine-tuning. The corresponding pretraining dataset can be taken from Figure 17. In the following the results on each of the four fine-tuning datasets are analyzed in detail.

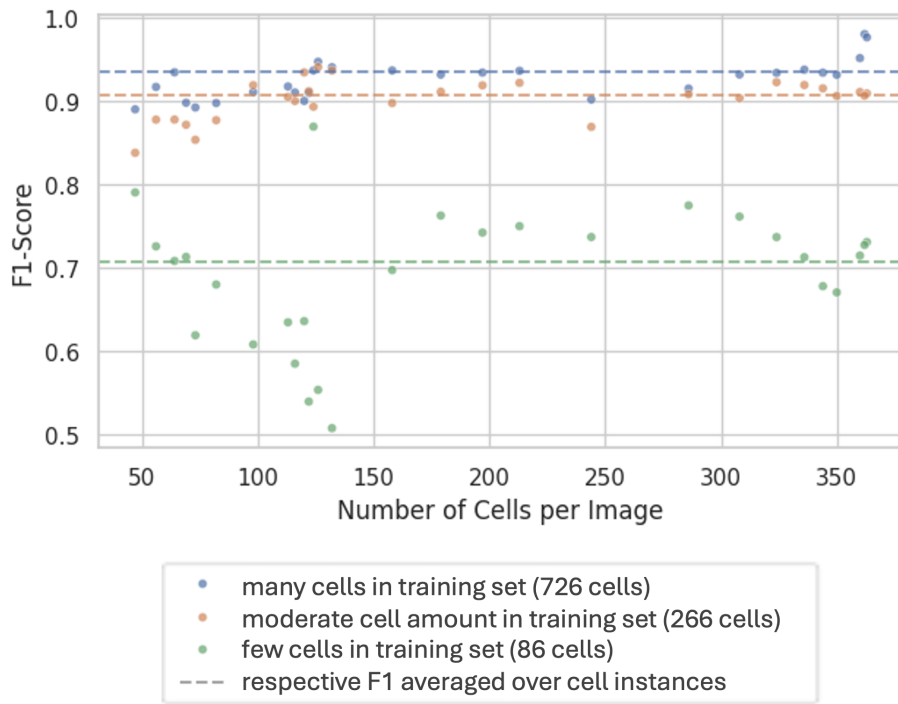


Figure 18: F1-score of each individual test set image from HELA over the number of cells present in the test set images. We compare three BNNs, each trained on two images with a total of 726 cells (blue), 266 cells (orange) and 86 cells (green).

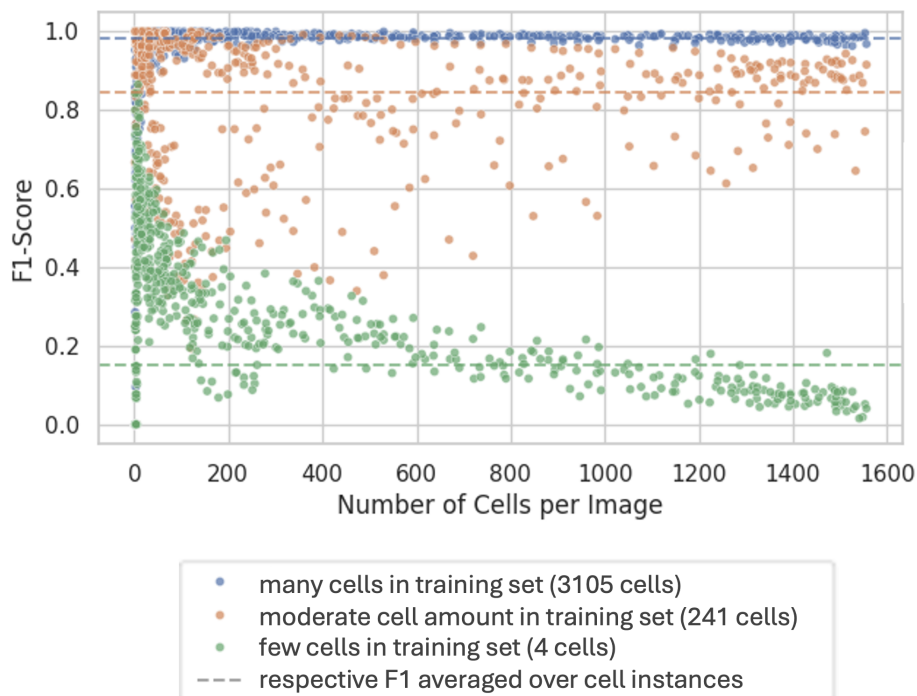


Figure 19: F1-score of each individual test set image from TOIAM over the number of cells present in the test set images. We compare three BNNs, each trained on two images with a total of 3105 cells (blue), 241 cells (orange) and 4 cells (green).

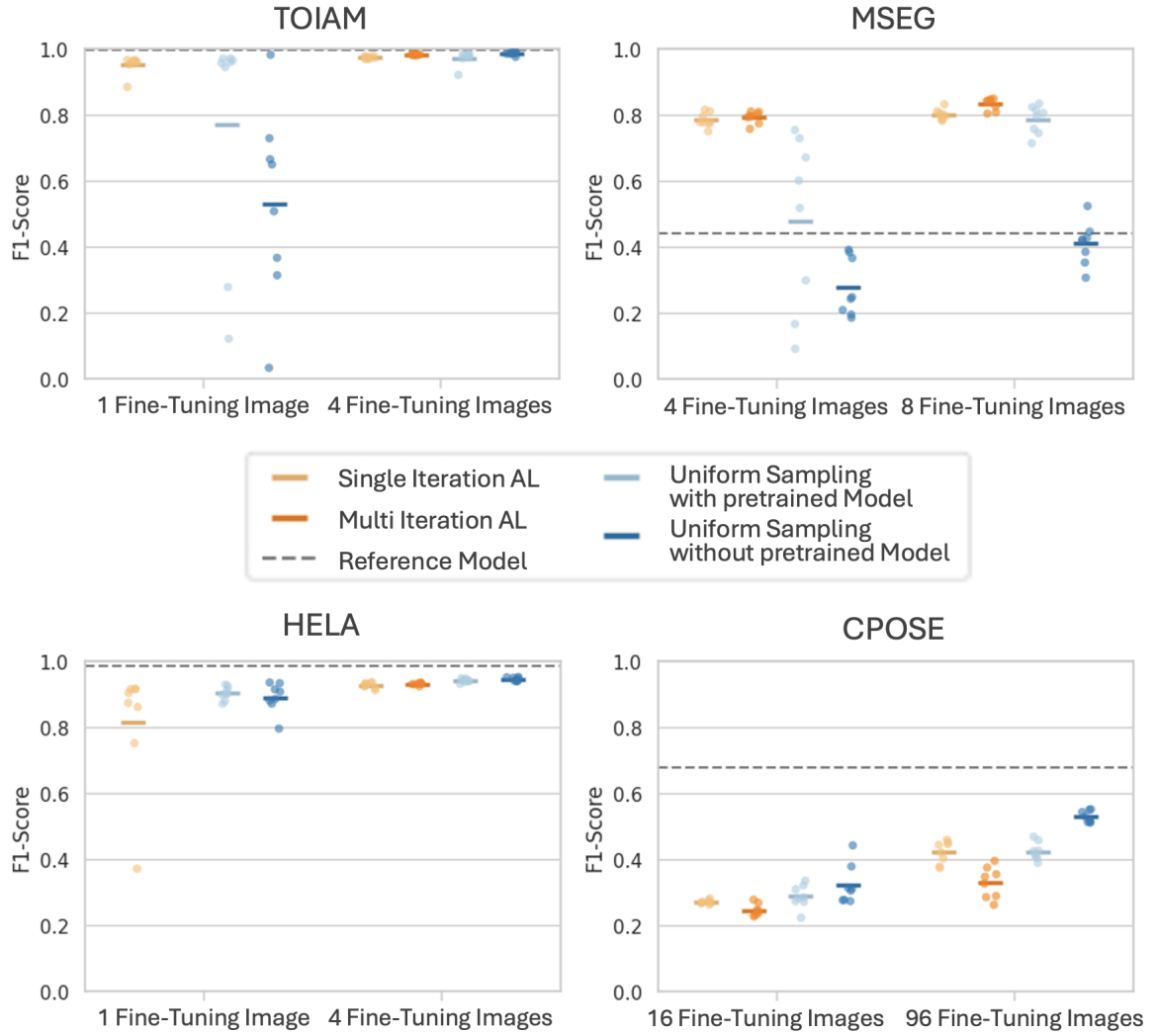


Figure 20: Segmentation performance on the test set of the fine-tuning dataset. The title of each chart is the dataset used for fine-tuning. The pretraining dataset can be taken from Figure 17. See section 4.2 for detailed explanation of the experiment design. Each point corresponds to the performance of one training run. The colored horizontal bars show the average F1-score.

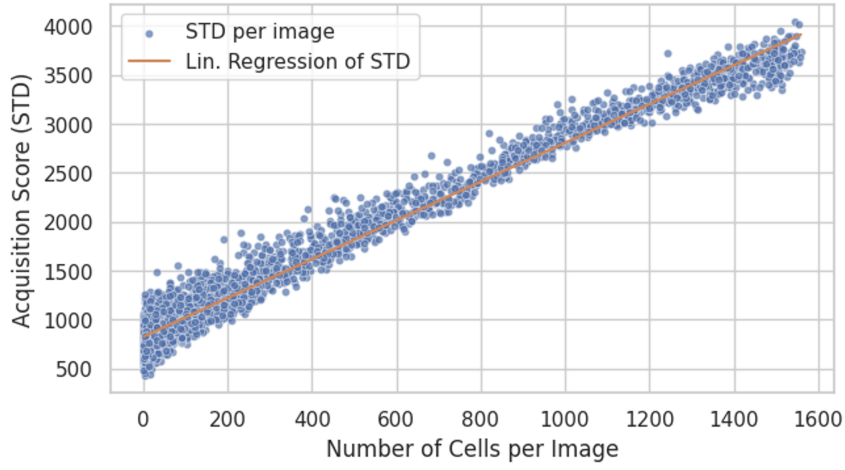


Figure 21: STD acquisition scores on the pool set of TOIAM, calculated from the uncertainty estimation of the pretrained BNN over the number of cells per image. The BNN was pretrained on MSEG.

Analysis of Fine-Tuning Results on TOIAM

As shown in Figure 19, even two training images with a moderate amount of cells are enough to have an F1 score above 0.8 on the test set. The models trained by four images even have an F1 score that is very close to the reference model that was trained on the whole training set. This may be due to the dataset being very homogeneous and adding more training images does not add new information. *"Uniform Sampling without pretrained Model"* has a lower average segmentation performance than the other models. This indicates that pretraining on MSEG improved the segmentation performance.

With only one training image, our AL method performs better than uniform sampling. Still, *"Uniform Sampling with pretrained Model"* also performs well in most cases except for two outliers. To explain this, we look at the acquisition score (STD) in dependency of the number of cells of an image shown in Figure 21. This strong correlation shows that the AL algorithm will always choose training images with a high number of cells. With our knowledge from section 6.4, we know that this leads to good performance. The two outliers of *"Uniform Sampling with pretrained Model"* were trained on an image containing only 2 and 4 cells respectively. Again, with the knowledge from section 6.4, this explains the bad segmentation performance.

Analysis of Fine-Tuning Results on MSEG

Here, active learning performs better than uniform sampling. To understand this better, we look at the uncertainty maps that the used pretrained model generates. Figure 22 shows the pixelwise standard deviation plotted for the four most uncertain images in the MSEG pool set from which the AL samples are drawn. The uncertainty is high where the brightness is high (see 1st image), where many cells are close to each other (see 2nd image), or at background areas close to cells (see 3rd image). The uncertainty is low at cells that are clearly separated (see left side of 4th image) and at background areas with less pixel variations. Intuitively this makes sense when we want uncertainty to correlate with informativeness. The more structural variations and densely packed cells are present in an image, the more likely it is to have "new" information for the model to learn from. Our method of measuring informativeness seems to follow this intuition, resulting in the good performance of AL.

Another result from Figure 20 is, that using the pretrained model is advantageous over learning with MSEG data from scratch. This is in line with the results from the experiments in section 6.3. Figure 16 shows that a model trained on CPOSE generalizes very well to MSEG data. By comparison to Figure 15, we see that the model trained on CPOSE performs even better on MSEG than the model trained on MSEG itself. This may be due to CPOSE being a

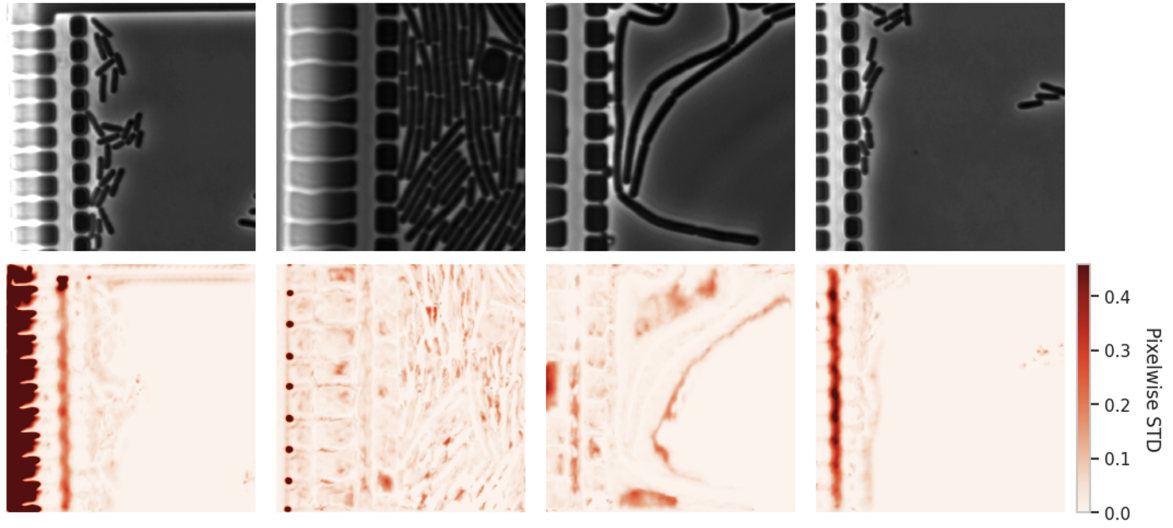


Figure 22: Four most uncertain images of the MSEG pool set. The first row shows the input image, the second row the corresponding uncertainty map. Uncertainties are generated by the model pretrained on CPOSE.

very heterogeneous and large dataset as compared to the much smaller MSEG dataset. [34] also observed that transfer learning worked particularly well, if the target dataset is small compared to the source dataset.

Also note that the performance increase between 4 and 8 used training images is very small for AL, but remarkably large for uniform sampling. When sampling images for AL in batches, the information gain may be lower for each additional image due to possible information overlap. It may be, that for the AL experiments the information overlap in the 8 training images is very large due to batch sampling.

Analysis of Fine-Tuning Results on HELA

Here, active learning performs worse than uniform sampling irrespective of the number of AL iterations or the number of fine-tuning images. Even "*Uniform Sampling without pretrained model*" performs better than AL on a pretrained model. This clearly shows, that the acquisition function does not sample good training images. To understand this behavior, we look at the uncertainty maps of the four most uncertain images in the MSEG pool set, shown in Figure 23. The uncertainty is high in the background and low around cells. Combining this knowledge with the results from Figure 18, it is clear that efficient learning through our AL method is not possible. This finding is strengthened by plotting the STD acquisition score for each image in the HELA dataset over the number of cells per image, shown in Figure 24. The acquisition function will always sample images with few cell, which contradicts our knowledge from section 6.4 that images with many cells are more useful for training.

The question arises, why the background is predicted to have such high uncertainties, while the input image has almost no pattern in the background. For a NN it should be easy to distinguish black background from bright objects. By looking closely at the uncertainty maps in Figure 23, we can see a grid pattern in the areas of high uncertainty, that is not present in the input image. This pattern is known as checkerboard artifacts and very common for U-Net architectures (see section 2.5). These artifacts may be part of the reason, why the background has such high uncertainties. How to overcome checkerboard artifacts is discussed in more detail in section 6.6.

To compare the performance on the HELA dataset in

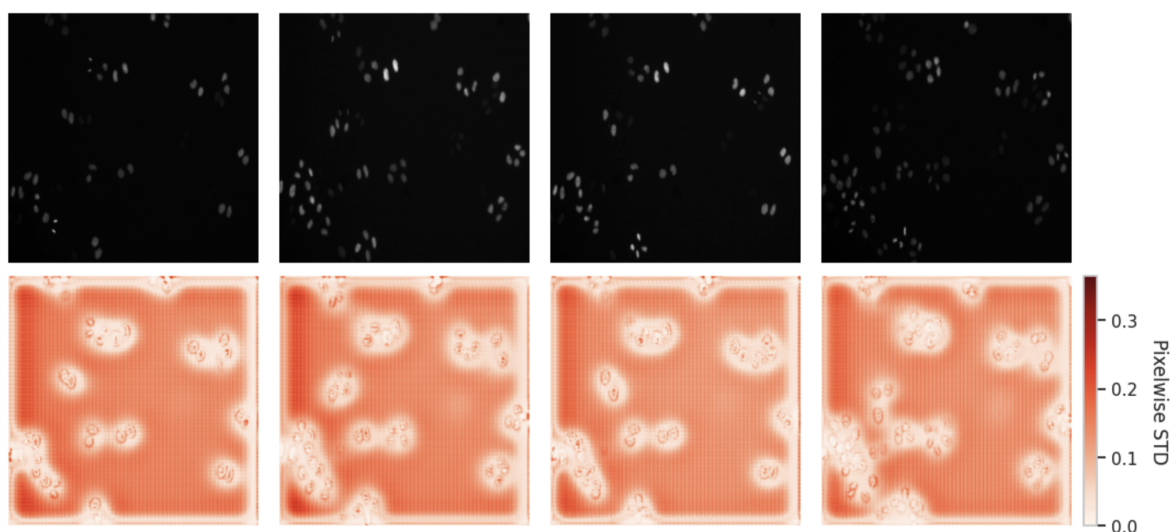


Figure 23: Four most uncertain images of the HELA pool set. The first row shows the input image, the second row the corresponding uncertainty map. Uncertainties are calculated by the model pretrained on TOIAM. The uncertainty maps display a grid-pattern in the background.

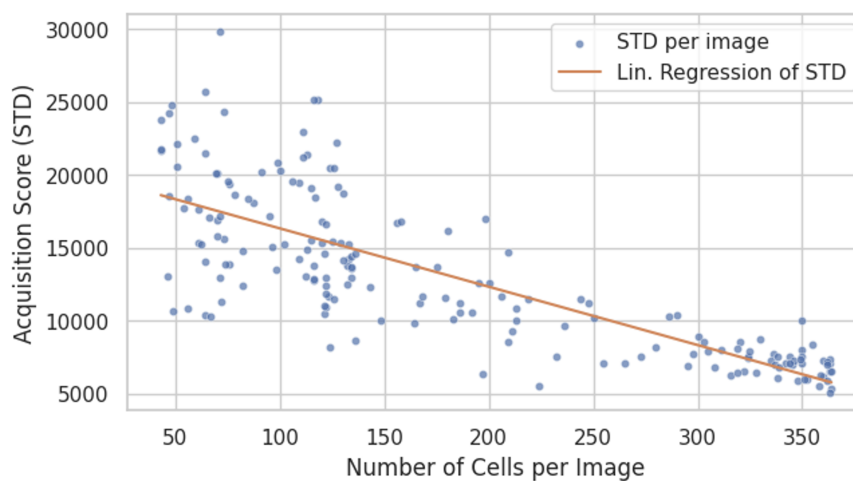


Figure 24: STD acquisition score on the pool set of HELA, calculated from the uncertainty estimation of the pretrained model over the number of cells per image. The model was pretrained on TOIAM.

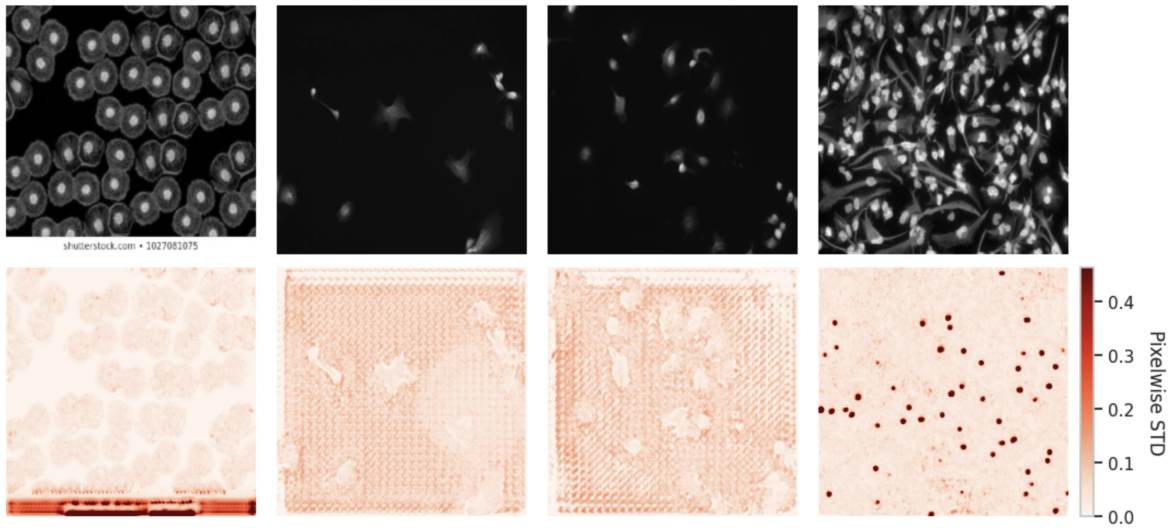


Figure 25: Four most uncertain images of the CPOSE pool set. The first row shows the input image, the second row the corresponding uncertainty map. Uncertainties are calculated by the model pretrained on HELA. The uncertainty map of the 2nd and 3rd image contain checkerboard artifacts.

Analysis of Fine-Tuning Results on CPOSE

Using the model pretrained on HELA performs very poorly on CPOSE. One reason can be that there is very little variation in the HELA dataset, whereas the type of microorganism present in CPOSE is very heterogeneous and strongly differs from HELA cells. During fine-tuning we want to find a local minima in the parameter space with respect to the performance score on the new data. We hope that this local minima is close to the parameters learned during pretraining, which is also why we reduce the learning rate. Here we seem to have the situation, that the pretraining dataset does not contain much information that can be reused in the fine-tuning step. A good local minima for the new dataset could not be found close to the initial pretrained parameters. It could be beneficial to use a different dataset for pretraining.

Active learning clearly underperforms compared to uniform sampling. By looking at the uncertainty maps of the most uncertain images in the pool set in Figure 25, we see checkerboard artifacts in the 2nd and 3rd image. These artifacts likely influence the acquisition score of the image, such that the uncertainty does not correlate with informativeness, which may explain the poor performance of AL. How to overcome checkerboard artifacts is discussed in the following.

6.6 Analysis of Checkerboard Artifacts

By manually looking through the predicted cell distance maps, we observed that all our BNs create checkerboard artifacts in their predictions with different intensity. These artifacts are then also visible in the uncertainty maps, that are generated from the predicted cell distance maps. If a more heterogeneous pretraining dataset (like MSEG or CPOSE) was used, the artifacts were less distinct. As discussed in section 2.5, checkerboard artifacts are common in CNN architectures that involve transpose convolutional layers, but also arise from convolutional layers during backpropagation. It is also known, that the network can learn to suppress the artifacts. This is likely what we observe when pretraining on MSEG or CPOSE. One of the suggested solutions is to use a kernel size divisible by the stride in the transpose convolutional layers [50]. This has already been incorporated in the architecture, shown in Table 3 and did not help to prevent the artifacts. Another proposed solution by [50] is to exchange all transpose convolutional layers with a resize-convolution. For our network architecture, this means that each of the transposed convolutional layers (Transposed Conv2D) in Table 3 has to be replaced by a resize operation followed by a convolutional layer. The detailed settings for these two transformations

Layer	Activation	Kernel	Stride	Padding
Resize (nearest neighbor)	no activation	-	-	zero padding
Conv2D	no activation	3x3	1	no padding

Table 12: Architectural changes with the aim to reduce checkerboard artifacts. Each transposed convolutional layer in Table 3 is replaced by the above two transformations. One image resizing operation with nearest neighbor interpolation followed by a convolutional layer.

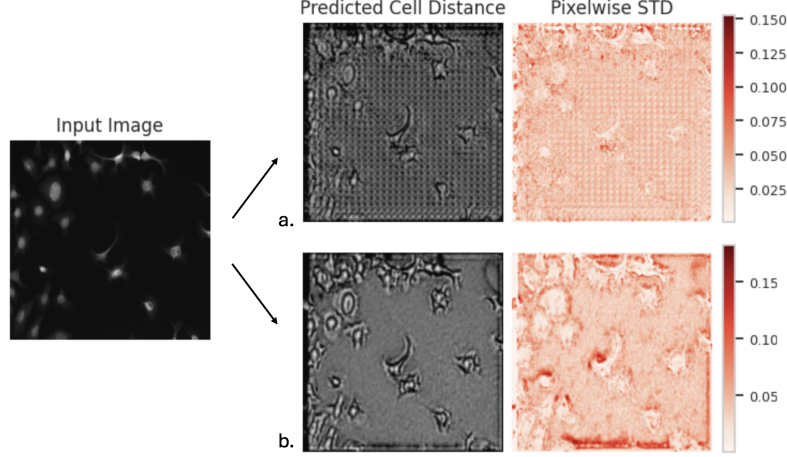


Figure 26: Predicted cell distance maps and uncertainty maps for a given input image of a BNN without (a.) and with (b.) the architectural changes to reduce checkerboard artifacts. For this input image, we observe that the checkerboard patterns are significantly reduced by the architectural changes.

are shown in Table 12.

Now we want to find out, if this architectural change is capable of reducing the checkerboard artifacts. Before the architectural change, we saw strong checkerboard artifacts in the predicted cell distance maps when training a BNN on HELA data and predicting on CPOSE data. An example prediction is shown in the upper row (a.) of Figure 26. When training a BNN with this architectural change on HELA, we did not see any checkerboard artifacts in the predictions by looking through the images manually. An example prediction is shown in the lower row (b.) of Figure 26.

Since we only reviewed the predictions qualitatively by looking at the cell distance maps, it may be that the artifacts are still present with different intensity in their pixel values or of different grid size. However, from the example picture we assume that the architectural change has a positive effect on reducing the checkerboard artifacts. Further analysis would be beneficial to quantify this effect.

Due to limited resources, we did not repeat the active learning experiments with the new architecture in this study. We would expect a change in the AL results where TOAIM and HELA were used as pretraining datasets, since those models were most effected by checkerboard artifacts.

7 Discussion

First, we found that the IOU as a segmentation performance measure is very sensitive to the hyperparameters of the post-processing algorithm. The thresholding method by Otsu [66] is not recommended in this case. We showed that finding the parameters via grid-search is a much better approach.

We also found that training models on the two rather homogeneous datasets TOIAM and HELA works very well. We achieve an average F1 score way above 0.8 on the test set with only two selected training images containing a moderate amount of cells. In case of TOIAM, this means with just two training images we can segment a dataset consisting of 4000 images. Further, we observed that the performance on TOIAM and HELA gets better the more cells are present in the training images.

Next, we discuss the results of our active learning experiments in section 6.5 for each of our used fine-tuning datasets. When fine-tuning our BNNs on **TOIAM**, we see that the average performance of our active learning method is higher than of uniform sampling, if one training image is used. However, "*Uniform Sampling with pretrained Model*" can also achieve high performance, but is less consistent over several training runs.

The active learning experiments on **MSEG** outperformed uniform sampling by a large margin. This suggests that AL with the uncertainty prediction from a diagonally Gaussian BNN samples images with a high content of information, leading to improved learning. From [27] we know that their method achieved an IOU (SEG) of 0.895 on the HELA dataset. Because their model was trained on a variety of different datasets, with only very few images from HELA, we compare it to our active learning method with 4 fine-tuning training images where we achieved an average IOU between 0.86 (Single Iteration AL) and 0.89 (Multi Iteration AL). Training a BNN on MSEG data without pretraining performs very bad compared to using CPOSE data for pretraining. The reason may be that CPOSE is a dataset that was particularly created for the purpose of pretraining. Also, MSEG is the smallest dataset with only 14 training images. This is in line with the literature, which also suggests that TL works best with a large pretraining dataset in combination with a small fine-tuning dataset [34].

Our active learning method did not outperform uniform sampling on the **HELA** dataset. As we could show in section 6.5, the used acquisition score did not correlate with the cell abundance of the images. However, from section 6.4 we know that this is necessary for good performance on this dataset.

For the **CPOSE** dataset, our active learning method has no benefit over uniformly sampling the training images. Here, our BNNs also suffer from the checkerboard artifacts, which may be one of the reasons for the poor performance compared to uniform sampling. Our BNNs achieved an IOU of 0.56 on the test set when trained with the whole training set (see Figure 15). [41] reported an IOU (Jaccard index) of about 0.63. Using HELA as pretraining dataset had no positive effect on the segmentation performance and is, in fact, a good example of negative transfer.

8 Conclusion

This thesis addresses the task of supervised instance segmentation with a combination of various machine learning techniques such as active learning, uncertainty estimation with Bayesian neural networks and transfer learning. The studied application, namely the segmentation of objects on microscopy images, holds several challenges for modern methods of artificial intelligence, such as little available training data, high-dimensional features and domain specific feature characteristics. It is therefore an ideal environment for studying highly specialized methods, that are targeting data efficient learning.

The research question postulated in section 1 could be positively answered for two out of four datasets. Using the uncertainty estimation of a diagonal Gaussian Bayesian neural network in an active learning framework led to improvements in segmentation performance over randomly sampling the training images of MSEG and TOIAM data. In case of the other two datasets, HELA and CPOSE, the Bayesian uncertainty estimations did not correlate well with informativeness. This leads to a lower segmentation accuracy of active learning compared to random sampling. One reason may be checkerboard artifacts in the predictions. We could show that this problem

can be reduced by a change in the network architecture. Whether this leads to a more data efficient learning through active learning needs to be evaluated in future work.

In addition to those findings, we also observed that choosing the hyperparameters of the watershed post-processing algorithm highly influences the IOU score and has to be done carefully. The solution to this is by using grid-search to find the hyperparameters. Further, using a suitable pretrained model that was fine-tuned to a new dataset via active learning was beneficial for three out of four datasets over training a randomly initialized model from scratch. This underlines the importance of transfer learning in fields where data is scarce.

Future research on this topic includes repeating the active learning experiments in section 6.5 with the improved network architecture from Table 12 to prevent checkerboard artifacts. Besides that, to get closer towards the goal of reducing the annotation effort, it is necessary to incorporate the annotation cost into the acquisition function. To estimate the annotation cost, the total cell boundary length of the predicted segmentation masks may be used. Algorithms for solving the 0-1 Knapsack problem could be used to integrate the annotation cost into the acquisition function used in this work.

References

- [1] A. Grünberger,
“Single-cell analysis of microbial production strains in microfluidic bioreactors,”
Ph.D. dissertation, RWTH Aachen University, Aachen, 2014.
- [2] S. Helfrich, “High-Throughput Live-Cell Imaging for Investigations of Cellular Heterogeneity in *Corynebacterium glutamicum*,”
Ph.D. dissertation, RWTH Aachen University, Aachen, 2016.
- [3] E. Upschulte, S. Harmeling, K. Amunts, and T. Dickscheid, “Contour proposal networks for biomedical instance segmentation,”
Medical Image Analysis, vol. 77, p. 102371, Jan. 2022, ISSN: 1361-8415.
DOI: 10.1016/j.media.2022.102371.
- [4] L. Jin, Z. Chen, and Z. Tu,
Object Detection Free Instance Segmentation With Labeling Transformations, Nov. 2016.
DOI: 10.48550/arXiv.1611.08991.
- [5] M. Herde, D. Huseljic, B. Sick, and A. Calma, “A Survey on Cost Types, Interaction Schemes, and Annotator Performance Models in Selection Algorithms for Active Learning in Classification,” *IEEE Access*, vol. 9, pp. 166 970–166 989, 2021, ISSN: 2169-3536.
DOI: 10.1109/ACCESS.2021.3135514.
- [6] Y. Gal and Z. Ghahramani,
Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference,
Jan. 2016. DOI: 10.48550/arXiv.1506.02158.
- [7] I. C. Saidu and L. Csató, “Active Learning with Bayesian UNet for Efficient Semantic Image Segmentation,” *Journal of Imaging*, vol. 7, no. 2, p. 37, Feb. 2021,
ISSN: 2313-433X. DOI: 10.3390/jimaging7020037.
- [8] L. Yang, Y. Zhang, J. Chen, S. Zhang, and D. Z. Chen, “Suggestive Annotation: A Deep Active Learning Framework for Biomedical Image Segmentation,”
in *Medical Image Computing and Computer Assisted Intervention MICCAI 2017*,
M. Descoteaux, L. Maier-Hein, A. Franz, P. Jannin, D. L. Collins, and S. Duchesne, Eds.,
Cham: Springer International Publishing, 2017, pp. 399–407, ISBN: 978-3-319-66179-7.
DOI: 10.1007/978-3-319-66179-7_46.
- [9] Y. Gal, R. Islam, and Z. Ghahramani, “Deep Bayesian active learning with image data,”
in *Proceedings of the 34th International Conference on Machine Learning*,
vol. 70, Sydney, Australia: Journal of Machine Learning Research, Aug. 2017,
pp. 1183–1192. DOI: 10.5555/3305381.3305504.
- [10] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra,
“Weight Uncertainty in Neural Networks,”
in *Proceedings of the 32nd International Conference on Machine Learning*,
vol. 37, Journal of Machine Learning Research, May 2015, pp. 1613–1622.
DOI: 10.5555/3045118.3045290.
- [11] A. Krogh and J. Hertz, “A Simple Weight Decay Can Improve Generalization,”
in *Advances in Neural Information Processing Systems*, vol. 4, Morgan-Kaufmann, 1991.

- [12] C. E. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, Jul. 1948, ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1948.tb01338.x.
- [13] D. J. C. MacKay, "A Practical Bayesian Framework for Backprop Networks," *Neural Computation*, vol. 4, no. 3, pp. 448–472, May 1992. DOI: 10.1162/neco.1992.4.3.448.
- [14] D. J. C. Mackay, "Probable networks and plausible predictions — a review of practical Bayesian methods for supervised neural networks," *Network: Computation in Neural Systems*, vol. 6, no. 3, pp. 469–505, 1995, ISSN: 0954-898X. DOI: 10.1088/0954-898X_6_3_011.
- [15] T. Salimans, D. Kingma, and M. Welling, "Markov Chain Monte Carlo and Variational Inference: Bridging the Gap," in *Proceedings of the 32nd International Conference on Machine Learning*, Jun. 2015, pp. 1218–1226.
- [16] F. Ruiz and M. Titsias, "A Contrastive Divergence for Combining Variational Inference and MCMC," in *Proceedings of the 36th International Conference on Machine Learning*, May 2019, pp. 5537–5545.
- [17] A. Graves, "Practical Variational Inference for Neural Networks," in *Advances in Neural Information Processing Systems*, vol. 24, Curran Associates, Inc., 2011, pp. 2348–2356. DOI: 10.5555/2986459.2986721.
- [18] N. Houlsby, F. Huszár, Z. Ghahramani, and M. Lengyel, *Bayesian Active Learning for Classification and Preference Learning*, Dec. 2011. DOI: 10.48550/arXiv.1112.5745.
- [19] B. Settles, "Active Learning Literature Survey," University of Wisconsin-Madison Department of Computer Sciences, Technical Report, 2009.
- [20] M. Gaillochet, C. Desrosiers, and H. Lombaert, "Active learning for medical image segmentation with stochastic batches," *Medical Image Analysis*, vol. 90, p. 102958, Dec. 2023, ISSN: 13618415. DOI: 10.1016/j.media.2023.102958.
- [21] A. Kirsch, J. van Amersfoort, and Y. Gal, "BatchBALD: Efficient and Diverse Batch Acquisition for Deep Bayesian Active Learning," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.
- [22] O. Sener and S. Savarese, *Active Learning for Convolutional Neural Networks: A Core-Set Approach*, Feb. 2018. DOI: 10.48550/arXiv.1708.00489.
- [23] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," University of Toronto, Computer Science Department, Technical Report.
- [24] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading Digits in Natural Images with Unsupervised Feature Learning," *Neural Information Processing Systems Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.
- [25] G. Citovsky, G. DeSalvo, C. Gentile, et al., "Batch Active Learning at Scale," in *Advances in Neural Information Processing Systems*, vol. 34, Curran Associates, Inc., 2021, pp. 11933–11944. DOI: 10.5555/3540261.3541174.

- [26] S. Mittal, J. Niemeijer, J. P. Schäfer, and T. Brox, "Best Practices in Active Learning for Semantic Segmentation," in *Pattern Recognition*, U. Köthe and C. Rother, Eds., Cham: Springer, 2024, pp. 427–442, ISBN: 978-3-031-54605-1. DOI: 10.1007/978-3-031-54605-1_28.
- [27] T. Scherr, K. Löffler, M. Böhlend, and R. Mikut, "Cell segmentation and tracking using CNN-based distance predictions and a graph-based matching strategy," *PLOS ONE*, vol. 15, no. 12, K. Metze, Ed., Dec. 2020, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0243219.
- [28] M. Gorriz, A. Carlier, E. Faure, and X. Giro-i-Nieto, *Cost-Effective Active Learning for Melanoma Segmentation*, Nov. 2017. DOI: 10.48550/arXiv.1711.09168.
- [29] W. Kuo, C. Häne, E. Yuh, P. Mukherjee, and J. Malik, "Cost-Sensitive Active Learning for Intracranial Hemorrhage Detection," in *Medical Image Computing and Computer Assisted Intervention 2018*, A. F. Frangi, J. A. Schnabel, C. Davatzikos, C. Alberola-López, and G. Fichtinger, Eds., Cham: Springer International Publishing, 2018, pp. 715–723, ISBN: 978-3-030-00931-1. DOI: 10.1007/978-3-030-00931-1_82.
- [30] B. Settles, M. Craven, and L. Friedland, "Active Learning with Real Annotation Costs," *Proceedings of the Neural Information Processing Systems Workshop on Cost-Sensitive Learning*, 2008.
- [31] S. Martello, D. Pisinger, and P. Toth, "Dynamic Programming and Strong Bounds for the 0-1 Knapsack Problem," *Management Science*, vol. 45, no. 3, pp. 414–424, 1999, ISSN: 0025-1909.
- [32] F. Zhuang, Z. Qi, K. Duan, *et al.*, "A Comprehensive Survey on Transfer Learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, Jan. 2021, ISSN: 1558-2256. DOI: 10.1109/JPROC.2020.3004555.
- [33] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" In *Advances in Neural Information Processing Systems*, vol. 27, Curran Associates, Inc., 2014.
- [34] D. Karimi, S. K. Warfield, and A. Gholipour, "Transfer learning in medical image segmentation: New insights from analysis of the dynamics of model parameters and learned representations," *Artificial Intelligence in Medicine*, vol. 116, p. 102 078, Jun. 2021, ISSN: 0933-3657. DOI: 10.1016/j.artmed.2021.102078.
- [35] M. Amiri, R. Brooks, and H. Rivaz, "Fine-Tuning U-Net for Ultrasound Image Segmentation: Different Layers, Different Outcomes," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 67, no. 12, pp. 2510–2518, Dec. 2020, ISSN: 1525-8955. DOI: 10.1109/TUFFC.2020.3015081.
- [36] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.
- [37] B. Serrano-Antón, A. Otero-Cacho, D. López-Otero, *et al.*, "Coronary Artery Segmentation Based on Transfer Learning and UNet Architecture on Computed Tomography Coronary Angiography Images," *IEEE Access*, vol. 11, pp. 75 484–75 496, 2023, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2023.3293090.

- [38] D. Cheng and E. Y. Lam, *Transfer Learning U-Net Deep Learning for Lung Ultrasound Segmentation*, Oct. 2021. DOI: 10.48550/arXiv.2110.02196.
- [39] B. Kaur, P. Lemaître, R. Mehta, *et al.*, "Improving Pathological Structure Segmentation via Transfer Learning Across Diseases," in *Domain Adaptation and Representation Transfer and Medical Image Learning with Less Labels and Imperfect Data*, Q. Wang, F. Milletari, H. V. Nguyen, *et al.*, Eds., Cham: Springer International Publishing, 2019, pp. 90–98, ISBN: 978-3-030-33391-1. DOI: 10.1007/978-3-030-33391-1_11.
- [40] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds., Cham: Springer International Publishing, 2015, pp. 234–241, ISBN: 978-3-319-24574-4. DOI: 10.1007/978-3-319-24574-4_28.
- [41] K. J. Cutler, C. Stringer, T. W. Lo, *et al.*, "Omnipose: A high-precision morphology-independent solution for bacterial cell segmentation," *Nature Methods*, vol. 19, no. 11, pp. 1438–1448, Nov. 2022, ISSN: 1548-7091, 1548-7105. DOI: 10.1038/s41592-022-01639-4.
- [42] R. J. Zaveri, V. Brume, and G. Doretto, *Few-shot adaptation for morphology-independent cell instance segmentation*, Feb. 2024. DOI: 10.48550/arXiv.2402.17165.
- [43] F. P. Kuhl and C. R. Giardina, "Elliptic Fourier features of a closed contour," *Computer Graphics and Image Processing*, vol. 18, no. 3, pp. 236–258, Mar. 1982, ISSN: 0146-664X. DOI: 10.1016/0146-664X(82)90034-X.
- [44] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," in *2017 IEEE International Conference on Computer Vision*, Venice: IEEE, Oct. 2017, pp. 2980–2988, ISBN: 978-1-5386-1032-9. DOI: 10.1109/ICCV.2017.322.
- [45] E. Upschulte, S. Harmeling, K. Amunts, and T. Dickscheid, "Uncertainty-Aware Contour Proposal Networks for Cell Segmentation in Multi-Modality High-Resolution Microscopy Images," in *Proceedings of The Cell Segmentation Challenge in Multi-modality High-Resolution Microscopy Images*, Proceedings of Machine Learning Research, Jun. 2023, pp. 1–12.
- [46] P. Naylor, M. Laé, F. Reyat, and T. Walter, "Segmentation of Nuclei in Histopathology Images by Deep Regression of the Distance Map," *IEEE Transactions on Medical Imaging*, vol. 38, no. 2, pp. 448–459, Feb. 2019, ISSN: 0278-0062, 1558-254X. DOI: 10.1109/TMI.2018.2865709.
- [47] S. Beucher and C. Lantuéjoul, "Use of Watersheds in Contour Detection," in *International Workshop on Image Processing, Real-time Edge and Motion Detection*, Sep. 1979.
- [48] A. Rosenfeld and J. L. Pfaltz, "Sequential Operations in Digital Picture Processing," *Journal of the Association of Computing Machinery*, vol. 13, no. 4, pp. 471–494, Oct. 1966, ISSN: 0004-5411. DOI: 10.1145/321356.321357.
- [49] R. M. Haralick, S. R. Sternberg, and X. Zhuang, "Image Analysis Using Mathematical Morphology," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, no. 4, pp. 532–550, Jul. 1987, ISSN: 0162-8828. DOI: 10.1109/TPAMI.1987.4767941.
- [50] A. Odena, V. Dumoulin, and C. Olah, "Deconvolution and Checkerboard Artifacts," *Distill*, Oct. 2016, ISSN: 2476-0757. DOI: 10.23915/distill.00003.

- [51] Y. Sugawara, S. Shiota, and H. Kiya, "Checkerboard artifacts free convolutional neural networks," *APSIPA Transactions on Signal and Information Processing*, vol. 8, e9, Jan. 2019, ISSN: 2048-7703. DOI: 10.1017/ATSIP.2019.2.
- [52] Z. Wojna, V. Ferrari, S. Guadarrama, *et al.*, "The Devil is in the Decoder: Classification, Regression and GANs," *International Journal of Computer Vision*, vol. 127, no. 11, pp. 1694–1706, Dec. 2019, ISSN: 1573-1405. DOI: 10.1007/s11263-019-01170-8.
- [53] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37, Lille, France: Journal of Machine Learning Research, Jul. 2015, pp. 448–456. DOI: 10.5555/3045118.3045167.
- [54] L. R. Dice, "Measures of the Amount of Ecologic Association Between Species," *Ecology, Ecological Society of America*, vol. 26, no. 3, pp. 297–302, 1945, ISSN: 1939-9170. DOI: 10.2307/1932409.
- [55] D. Mahapatra, B. Bozorgtabar, J.-P. Thiran, and M. Reyes, "Efficient Active Learning for Image Classification and Segmentation Using a Sample Selection and Conditional Generative Adversarial Network," in *Medical Image Computing and Computer Assisted Intervention 2018*, A. F. Frangi, J. A. Schnabel, C. Davatzikos, C. Alberola-López, and G. Fichtinger, Eds., Cham, Switzerland: Springer International Publishing, 2018, pp. 580–588, ISBN: 978-3-030-00934-2. DOI: 10.1007/978-3-030-00934-2_65.
- [56] M. Kozubek, *Cell Tracking Challenge*. [Online]. Available: <https://celltrackingchallenge.net/> (visited on 09/14/2024).
- [57] P. Matula, M. Maška, D. V. Sorokin, P. Matula, C. Ortiz-de-Solórzano, and M. Kozubek, "Cell Tracking Accuracy Measurement Based on Comparison of Acyclic Oriented Graphs," *PLOS ONE*, vol. 10, no. 12, Dec. 2015, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0144959.
- [58] C. Stringer, T. Wang, M. Michaelos, and M. Pachitariu, "Cellpose: A generalist algorithm for cellular segmentation," *Nature Methods*, vol. 18, no. 1, pp. 100–106, Jan. 2021, ISSN: 1548-7091, 1548-7105. DOI: 10.1038/s41592-020-01018-x.
- [59] R. F. Laine, I. Arganda-Carreras, R. Henriques, and G. Jacquemet, "Avoiding a replication crisis in deep-learning-based bioimage analysis," *Nature Methods*, vol. 18, no. 10, pp. 1136–1144, Oct. 2021, ISSN: 1548-7105. DOI: 10.1038/s41592-021-01284-3.
- [60] J. Seiffarth, L. Blöbaum, K. Löffler, *et al.*, *Data for - Tracking one in a million: Performance of automated tracking on a large-scale microbial data set*, Oct. 2022. DOI: 10.5281/ZENODO.7260137.
- [61] T. Scherr, J. Seiffarth, B. Wollenhaupt, *et al.*, *microbeSEG dataset*, Apr. 2022. DOI: 10.5281/ZENODO.6497715.
- [62] B. Neumann, T. Walter, J.-K. Hériché, *et al.*, "Phenotypic profiling of the human genome by time-lapse microscopy reveals cell division genes," *Nature*, vol. 464, no. 7289, pp. 721–727, Apr. 2010, ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature08869.
- [63] C. Stringer and M. Pachitariu, *Cellpose*. [Online]. Available: <http://www.cellpose.org> (visited on 09/14/2024).

- [64] D. D. Polsby and R. D. Popper, "The Third Criterion: Compactness as a Procedural Safeguard against Partisan Gerrymandering,"
Yale Law & Policy Review, vol. 9, no. 2, pp. 301–353, 1991, ISSN: 0740-8048.
- [65] D. W. Scott, "On optimal and data-based histograms,"
Biometrika, vol. 66, no. 3, pp. 605–610, Dec. 1979, ISSN: 0006-3444.
DOI: 10.1093/biomet/66.3.605.
- [66] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms,"
IEEE Transactions on Systems, Man, and Cybernetics, vol. 9, no. 1, pp. 62–66, Jan. 1979,
ISSN: 2168-2909. DOI: 10.1109/TSMC.1979.4310076.
- [67] S. Kullback and R. A. Leibler, "On Information and Sufficiency,"
The Annals of Mathematical Statistics, vol. 22, no. 1, pp. 79–86, Mar. 1951,
ISSN: 0003-4851, 2168-8990. DOI: 10.1214/aoms/1177729694.

Appendix A

Propositions and Proofs

Equivalence of KL-divergence and VFE minimization

Proof. Equivalence of finding the minimizer of the KL-divergence and the variational free energy can be shown, by first applying the definition of the KL-divergence [67].

$$\begin{aligned}\theta^* &= \arg \min_{\theta} \text{KL} [q(w|\theta) || P(w|D)] \\ &= \arg \min_{\theta} \int_{-\infty}^{\infty} q(w|\theta) \log \frac{q(w|\theta)}{P(w|D)} dw\end{aligned}$$

Then we replace the posterior by the prior times the likelihood and reformulate the expression such that the prior and the likelihood are written as separate terms.

$$\begin{aligned}&= \arg \min_{\theta} \int_{-\infty}^{\infty} q(w|\theta) \left(\log \frac{q(w|\theta)}{P(w)} - \log P(D|w) \right) dw \\ &= \arg \min_{\theta} \int_{-\infty}^{\infty} q(w|\theta) \log \frac{q(w|\theta)}{P(w)} - q(w|\theta) \log P(D|w) dw\end{aligned}$$

Which further gives us the KL-divergence of the approximate posterior and the prior subtracted by the expectation of the log likelihood.

$$\begin{aligned}&= \arg \min_{\theta} \text{KL} [q(w|\theta) || P(w)] - \int_{-\infty}^{\infty} q(w|\theta) \log P(D|w) dw \\ &= \arg \min_{\theta} \text{KL} [q(w|\theta) || P(w)] - \mathbb{E}_{q(w|\theta)} [\log P(\mathcal{D}|w)] \\ &= \arg \min_{\theta} \mathcal{F}(\mathcal{D}, \theta)\end{aligned}$$

Thus, we arrive at the VFE. [10] □

Approximating the variational free energy

Proof. Intuition for the approximation of the VFE by [10]. First the VFE needs to be reformulated as

$$\begin{aligned}\mathcal{F}(\mathcal{D}, \theta) &= \text{KL} [q(w|\theta) || P(w)] - \mathbb{E}_{q(w|\theta)} [\log P(\mathcal{D}|w)] \\ &= \int_{-\infty}^{\infty} q(w|\theta) (\log q(w|\theta) - \log P(w)) dw - \int_{-\infty}^{\infty} q(w|\theta) \log P(D|w) dw \\ &= \int_{-\infty}^{\infty} q(w|\theta) * [\log q(w|\theta) - \log P(w) - \log P(D|w)] dw \\ &= \mathbb{E}_{q(w|\theta)} [\log q(w|\theta) - \log P(w) - \log P(D|w)]\end{aligned}$$

We can now approximate the expectation value by Monte Carlo integration.

$$\mathcal{F}(\mathcal{D}, \theta) \approx \sum_{i=1}^n \log q(w^{(i)}|\theta) - \log P(w^{(i)}) - \log P(D|w^{(i)})$$

with $w^{(i)} \sim q(w|\theta)$ being acquired by Monte Carlo sampling and thus we arrive at the approximated VFE proposed by [10]. $\square$

Proposition used for calculating the gradients of θ

This proposition is taken from [10].

Proposition 1. *Let ϵ be a random variable having a probability density given by $q(\epsilon)$ and let $w = t(\theta, \epsilon)$ where $t(\theta, \epsilon)$ is a deterministic function. Suppose further that the marginal probability density of w , $q(w|\theta)$, is such that $q(\epsilon)d\epsilon = q(w|\theta)dw$. Then for a function f with derivatives in w :*

$$\frac{\partial}{\partial \theta} \mathbb{E}_{q(w|\theta)} [f(w, \theta)] = \mathbb{E}_{q(\epsilon)} \left[\frac{\partial f(w, \theta)}{\partial w} \frac{\partial w}{\partial \theta} + \frac{\partial f(w, \theta)}{\partial \theta} \right].$$

Proof. We start by expressing the expectation as an integral and apply the equation for $q(w|\theta)dw$ from the proposition.

$$\begin{aligned} \frac{\partial}{\partial \theta} \mathbb{E}_{q(w|\theta)} [f(w, \theta)] &= \frac{\partial}{\partial \theta} \int f(w, \theta) q(w|\theta) dw \\ &= \frac{\partial}{\partial \theta} \int f(w, \theta) q(\epsilon) d\epsilon \end{aligned}$$

Because $q(\epsilon)$ is independent of θ , we can apply the Leibniz integral rule to arrive at the following.

$$= \int \frac{\partial f(w, \theta)}{\partial \theta} q(\epsilon) d\epsilon$$

Finally by applying the chain rule and because we have $w = t(\theta, \epsilon)$ from the proposition, we can rewrite the integral and express it as the expectation from above. [10]

$$\begin{aligned} &= \int \frac{\partial f(w, \theta)}{\partial w} \frac{\partial w}{\partial \theta} + \frac{\partial f(w, \theta)}{\partial \theta} q(\epsilon) d\epsilon \\ &= \mathbb{E}_{\epsilon \sim q(\epsilon)} \left[\frac{\partial f(w, \theta)}{\partial w} \frac{\partial w}{\partial \theta} + \frac{\partial f(w, \theta)}{\partial \theta} \right] \end{aligned}$$

$\square$