Evaluating the Applicability of Transfer Learning for Deep Learning Based Segmentation of Microscope Images

Bachelor's Thesis

submitted by

Benjamin Wilhelm

at the

Universität Konstanz

Faculty of Sciences

Department of Computer and Information Science

1. Reviewer: Prof. Dr. Bastian Goldlücke

2. Reviewer: Prof. Dr. Oliver Deussen

Konstanz, 2019

Benjamin Wilhelm
Evaluating the Applicability of Transfer Learning for Deep Learning Based Segmentation of Microscope Images
Bachelor's Thesis, University of Konstanz, 2019.

Abstract

Instance segmentation is a valuable preprocessing step for many analysis workflows of microscope image data. New deep learning based methods have surpassed classical methods. However, they often require a large amount of labeled training data and deep knowledge about the model. Therefore, they are more complicated to apply to new data.

This thesis investigates approaches to surpass these drawbacks. Initializing new models with pretrained weights for transfer learning is a promising technique to reduce the amount of required training data. Therefore, the applicability of transfer learning on microscope images is evaluated. The evaluation shows that transfer learning can improve the performance of models with only a few training examples and reduce the training time drastically. However, the choice of the pretraining dataset is crucial. The pretraining dataset needs to be diverse but related to the target dataset.

Additionally, this thesis introduces a deep learning segmentation framework within KNIME Analytics Platform to address the drawback of required expert knowledge. The framework hides most of the complexity for non-computer-scientists and makes it easy to use state-of-the-art models for instance segmentation. At the same time, the framework is easily extensible by more experienced users. The developed framework can handle large images by applying deep learning models on tiles of the image, reducing the required memory.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Today, modern microscopes can quickly produce a vast amount of image data from one or multiple experiments [Eco+16; Hui04; Kel+11; Ste+14]. The images have to be analyzed to draw conclusions from the experiments. Manual analysis can be time-consuming, tedious, and often does not yield convincing evidence because the scientist might have introduced a bias towards his assumptions [Wik19a]. Not to mention, that often the amount of data is just too large to be analyzed manually. Therefore an automatic analysis is desired. Using image analysis tools to process the data, scientists can produce reliable numeric results that proof (or disproof) their hypothesis and build a reproducible analysis pipeline which can be reused by other scientists.

One crucial preprocessing step for many automatic analysis pipelines is instance segmentation (in particular cell segmentation [Mei12]) that assigns each pixel of an image a class label and an instance id. The resulting segmentation can be used for further analysis, like intensity and shape measurements [BHL15; DB16] and tracking [Mei+09].

Recently, deep learning based methods for instance segmentation have emerged. Some methods have been introduced for natural images [He+17; HHS17] but can also be used on microscope images. Others have been introduced specifically for microscopic image analysis [RFB15; Sch+18]. StarDist [Sch+18] is a recent method that proved to be very powerful for segmenting objects with roundish shapes and will be the primary method used in this thesis. These methods can be used to segment datasets that are too difficult for traditional approaches.

However, there are downsides to these deep learning based methods. Firstly, they usually require an expert to build, train, and run a model for a specific experiment. Many methods require programming knowledge and a deep understanding of the model to use them. If the method should be part of an entire analysis pipeline, either the complete analysis requires programming knowledge, or the results must be transferred to another tool to do the other parts of the analysis. Using multiple tools in one analysis pipeline adds complexity to the pipeline and makes it harder to rerun it, change it, and to ensure its reproducibility. Deep learning based segmentation methods would be more useful if they were integrated into the tools that are already used for the analysis. They should be integrated in such a way that they can be used without in-depth knowledge

about the method itself.

Secondly, deep learning based methods usually require many labeled training examples to achieve good performance. Acquiring labeled data is often cumbersome and labor-intensive. Reducing the amount of data that is needed to train a deep learning model would make it more useful.

Contributions This work addresses the two downsides mentioned above of deep learning based methods. Transfer learning is a simple approach to reduce the amount of required training data. Therefore, the applicability of transfer learning to microscopy images is evaluated. Different scenarios with pretraining on simulated images, real microscope images, and real natural images are analyzed and discussed.

A framework for segmentation in KNIME Analytics Platform is introduced to enable non-experts to train and apply deep learning based segmentation methods. The framework is easy to use, extensible, can be used for transfer learning, and can process large images.

Additionally, the StarDist model is improved, to enable the segmentation of more challenging datasets, by building the U-Net backbone with residual bottleneck blocks. The proposed changes are evaluated and do improve the performance of the model on complex datasets significantly.

Structure In Chapter 2, an introduction to the essential foundations is given. Afterward, the related work is analyzed in Chapter 3. The StarDist [Sch+18] method and the proposed improvement are described in Chapter 4. In Chapter 5, transfer learning is evaluated to reduce the number of required training examples. The acquired insights are applied in Chapter 6, where the KNIME Instance Segmentation framework is described. A conclusion is drawn in Chapter 7.

Chapter 2

Foundations

2.1 Microscope Images

"Microscopy [...] has served as a fundamental scientific technique for centuries. Indeed, for hundreds of years, it was arguably the *only* scientific method. It remains an invaluable tool in biology and healthcare and has been integrated increasingly into modern chemical instrumentation." [BM09].

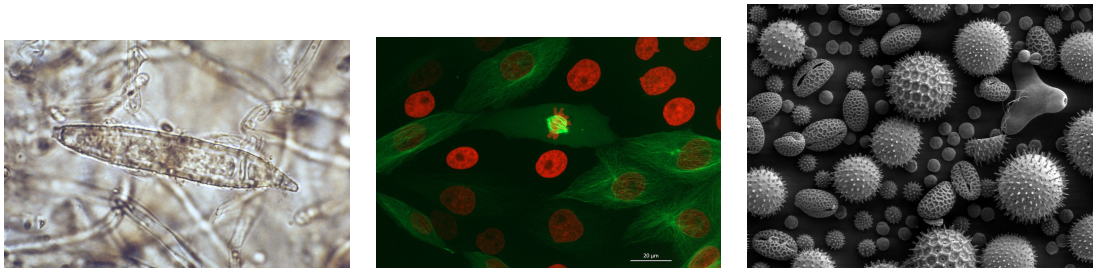
Automatic analysis of microscope image is now more critical than ever because modern microscopes can produce vast amounts of data [Eco+16; Hui04; Kel+11; Ste+14]. Before we focus on the important analysis task of instance segmentation, we will quickly discuss the different types of microscopes and image acquisitions to gain a better understanding of the acquired images.

2.1.1 Types of Microscopes

Microscopes can be categorized based on what interacts with the sample to generate the image. Light interacts with the sample for optical microscopy, electrons for electron microscopy, or a probe for scanning probe microscopy [Wik19c].

Optical Microscopes For optical microscopy, the microscope observes the interaction between light and the matter of the sample. There are different techniques of how light interacts with the sample and what is perceived by the sensor. These techniques apply to different kinds of samples and produce fundamentally different images.

The most straightforward form of optical microscopy is bright field microscopy where a sample gets illuminated from below, and the shadow of the sample is observed [BM09] (see Figure 2.1a). Dark field microscopy illuminates the sample such that light gets reflected from the sample [BM09]; phase contrast microscopy observes the phase shift of the light transmitted through the sample [BM09]; in differential interference contrast microscopy, the difference between two light beams is observed. For fluorescence microscopy, the effect of fluorescence is used by



(a) *Microsporum audouinii macroconidium* in bright field microscopy. Image by R. Summerbell and J. Scott [SS13].

(b) LLC-PK1 cells using fluorescence microscopy. Image by ZEISS Microscopy from Germany [ZEI15].

(c) Pollen from different plants using electron microscopy. Image by Dartmouth College Electron Microscope Facility [Dar].

Figure 2.1: Images taken by different types of microscopes.

illuminating the sample with high energy light such that it starts emitting light of longer wavelength [Wik19b] (see Figure 2.1b).

Diffraction limits the resolution of optical microscopy to around 250 nm [Wik19c].

Electron Microscopes To take higher resolution images than possible with light an electron beam (shorter wavelength than light) is used in electron microscopy [Wik19c] (see Figure 2.1c).

Scanning Probe Microscopes A physical probe that scans the surface of the sample enables atomic-level resolution for scanning probe microscopy [Wik19c].

This work will focus on images of optical microscopy. The experiments use fluorescence images and bright field images.

2.1.2 Image Acquisition

Sensors identical (or similar) to those in conventional digital cameras a are used for the image acquisition in microscopes [BM09]. The difference is that some microscopes can capture 3-dimensional images and several channels that is not three. In fluorescence microscopy, for example, there can be different stains (depending on the application) which all produce one individual channel.

Therefore we formally define an image with an arbitrary number of dimensions and channels:

Definition 2.1 (Image)**.** *An n-dimensional image f with c channels is a function which maps from a domain $\Omega \subset \mathbb{R}^n$ to a tuple of intensity value: $f : \Omega \to \mathbb{R}^c$.*

In this work, we will use 2-dimensional images with one channel which are sampled on integer coordinates. Therefore we will use $f : \Omega \subset \mathbb{Z}^2 \to \mathbb{R}$ from now on.

(a) Kittens (b) Semantic segmentation (c) Instance segmentation

Figure 2.2: Semantic segmentation and instance segmentation of kittens. (Image by "axelle b" [axe])

2.2 Instance Segmentation

A critical preprocessing step that allows for different kinds of automated analyses of microscopy image data is instance segmentation. An instance segmentation helps to locate various objects in an image and separate different instances of the same object.

Before we define an instance segmentation, we will define a general segmentation of an image.

Definition 2.2 (Segmentation)**.** *If L is a set of labels, a segmentation $s_f : \Omega \to L$ of an image $f : \Omega \to \mathbb{R}^c$ assigns each point in the image a label that describes the point in the original image f.*

The definition of segmentation does not define what the labels are. Neither does it define how to decide if a point in an image belongs to a specific label. There are two main options for choosing labels. The first option is to use semantic classes of the content of the image.

Definition 2.3 (Semantic Segmentation)**.** *A semantic segmentation is a segmentation where the labels L describe the class of the point in the image.*

Note that in a semantic segmentation the same label is assigned to different objects of the same class.

The second option is to use the class and object instance.

Definition 2.4 (Instance Segmentation)**.** *An instance segmentation is a segmentation where the labels L describe the class and the object instance of a point in the image.*

Figure 2.2 shows the difference between a semantic segmentation and an instance segmentation.

In this work, we will focus on instance segmentation with the only classes being foreground and background (We will use the term *segmentation* for them). Therefore our segmentations will have the labels $L \subset \mathbb{N}$ where 0 is used for background points, and all other numbers identify instances of foreground objects.

Applications An instance segmentation of cells can be used to extract statistics about the cells like size and shape or texture. They allow researchers to compare cells against each other (e.g., after some cells had been treated with a potential active substance). The extracted values can be evaluated yielding statistically significant statements.

Another application where an instance segmentation is a valuable preprocessing step is tracking. An instance segmentation can be used to track cells through images and detect divisions [Ren+15; Ulm+17] which results in a developmental lineage tree of the organism. The lineage tree and the tracking result are valuable information for biologists to understand how cell tissues develop into complex organisms. Note, that one can also omit the pixel segmentation and do tracking by detection (using the center points of each segment). However, the segmentation might contain valuable information that could improve the tracking. For example, the size or shape of objects is often not likely to change between frames.

2.3 Deep Learning

Many recent methods for segmentation make use of deep learning [He+17; RFB15; Sch+18]. Deep convolutional networks have been used for many computer vision tasks like image classification [KSH12; SZ15; He+16] and object detection [Ser+14; Gir+14; Red+16]. This section will briefly introduce the basic building blocks of deep neural networks.

2.3.1 Multilayer Perceptron

Perceptrons are the basis of each neural network. Perceptrons are units that have n inputs $\mathbf{x} = (x_1, \ldots, x_n)^T$ and one output. The parameters of a perceptron are a weight vector $\mathbf{w} = (w_1, \ldots, w_n)^T$ and a bias value b. The output of a perceptron is computed by applying an activation function to $\mathbf{w}^T\mathbf{x} + b$. The activation function of a perceptron is a step function and 1 if $\mathbf{w}^T\mathbf{x} + b$ larger than 0 and 0 else.

Multiple perceptrons can be combined in a feed-forward fashion (the inputs of *layers* of perceptrons are only connected to the outputs of the previous layer) to form a multilayer perceptron. A multilayer perceptron can be optimized for some data via the Backpropagation Algorithm which computes the derivative of a loss function for each weight. However, it is only possible to compute the derivative for each weight if the activation functions of the perceptrons are differentiable. The step function that is used for a single perceptron is not differentiable, and another function has to be used. We also call a perceptron with an arbitrary activation function a neuron. An activation function that can be used is the sigmoid function $f(x) = 1/(1 + e^{-x})$, which is an S-like curve between 0 and 1. The ReLU [NH10] function $f(x) = x$ if $x > 0$ and $f(x) = 0$ else is not differentiable at every point but still has been proven to be a good activation function for multilayer perceptrons. The model parameters can be optimized using statistical gradient descent or another optimizer like Adam [KB15].

Note that a multilayer perceptron is just a feed-forward neural network. There are also recurrent neural networks that will not be covered in this thesis.

2.3.2 Deep Neural Networks

Convolutional Neural Networks

Connecting all pixels of an image to some neurons would result in many weights that have to be learned. Additionally, if visual entities would move slightly in an image, they would not be recognized anymore. Convolutional neural networks have been proposed [LeC+99] to tackle this issue. In convolutional neural networks, some layers of neurons are replaced by convolutions with learned kernels. Note, that this is just a limitation on fully-connected layers by sharing weights and setting many weights to zero. Convolutional layers can learn useful features on images with far fewer parameters and are shift-invariant.

Sub-sampling layers are used between convolutional layers to reduce the size of the image. A usual sub-sampling strategy is max-pooling which only selects the maximum values of a block of values.

Residual Learning

Increasing the number of layers of neural networks improves the accuracy up to a point where it starts to degrade rapidly [He+16]. The accuracy does not degrade because of overfitting, but because the deep models are not as easy to optimize.

Residual learning [He+16] was introduced to solve this problem. A residual building block adds a skip connection that goes around some stacked layers. The input of the layers is added to the output of the layers. This way, if the output of the stacked layer is zero, the whole block maps to the identity. Therefore it is easy for a residual block to learn mappings that are close to the identity mapping. It was shown that this helps with the optimization of very deep neural networks [He+16].

2.4 Transfer Learning

It is much easier for a human to learn driving a truck if he learns how to drive a car beforehand. He will automatically make use of the knowledge of the task of driving a car and transfer it to the more complex task of how to drive a truck. Humans use this kind of transfer learning daily, but machine learning models are usually trained from scratch [Zha+19].

Transfer learning means adapting knowledge from one task to another task. The idea is that a model can reuse knowledge from the first task to easier learn how to solve the second task. This is only the case if some knowledge from the first task applies to the second task. If knowledge can be reused, the model might be able to learn the second task quicker and with less training examples.

For many tasks, there is not enough labeled data to train a model from scratch and manually labeling enough data is tedious, labor-intensive, and error-prone. On the other hand, it is likely that a dataset with a sufficient amount of data that is similar to the desired data exists. Transfer learning can be used to transfer learned knowledge from a model trained on the large dataset to the smaller dataset. This can enable training models without the need to label much data.

When transfer learning a model, the domain (kind and distribution of input data), the task (assignment of output labels), or both can change from the source to the target.

Definition 2.5 (Domain [PY10])**.** *A domain $\mathcal{D}$ consists of a feature space $\mathcal{X}$ and a probability distribution $P(\mathbf{x})$ with $\mathbf{x} \in \mathcal{X}$.*

Definition 2.6 (Task [PY10])**.** *Given a domain, $\mathcal{D} = (\mathcal{X}, P(\mathbf{x}))$, a task $\mathcal{T}$ consists of a label space $\mathcal{Y}$ and an objective function $f : \mathcal{X} \to \mathcal{Y}$.*

Definition 2.7 (Transfer Learning)**.** *Given a source domain $\mathcal{D}_S$ and task $\mathcal{T}_S$ and a target domain $\mathcal{D}_T$ and task $\mathcal{T}_T$, transfer learning aims to improve a model on the target task $\mathcal{T}_T$ using knowledge of the source task $\mathcal{T}_S$.*

2.4.1 Deep Transfer Learning

The usual approach to do transfer learning of deep neural networks is to train a model for the source task and copy weights from this source model to the target model. These weights are expected to compute valuable features also on the data of the target task. One has to decide which weights should be copied and how the target model should look like.

The two usual approaches are to copy most of the weights and fine-tune them on the target task or to apply a new model (logistic regression, SVM) to some high-level features of the source model.

Model on high-level features Training a new model to some high-level features (activations of a late layer) of the source model can allow creating a model for the target task very quickly. The resulting target model consists of the first part of the source model where the weights will not be adapted, and the new model on the high-level features. See model M_{T2} in Figure 2.3.

For this option, one has to decide which layer the activations should be taken from. Usually, one of the last layers will be used because a less powerful model on top of the activations will only be able to model easy relationships.

This option is relatively cheap because the new model is usually much easier to train than a full deep learning model. Also, if the new model does have a lower capacity, it will not overfit on small datasets.

Fine-tuning For fine-tuning, the same model as for the source task will be used, but the weights up to a certain layer will be initialized with the corresponding weights

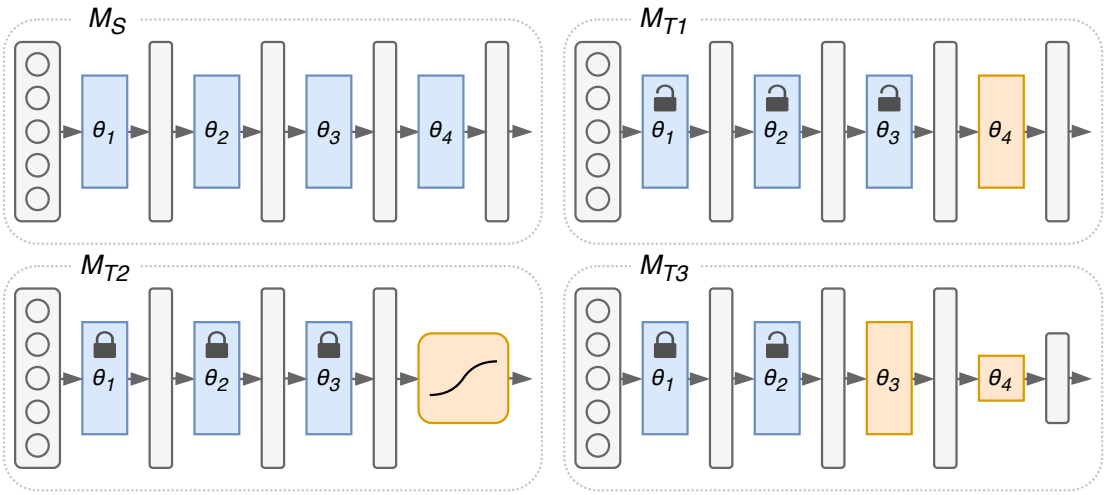


Figure 2.3: Illustration of how weights of deep learning models can be reused in other models for transfer learning. The weights of the source model M_S are marked blue while random weights are marked orange. The transferred weights can be fine-tuned (marked by an open lock) or fixed (marked by a closed lock) during the training of the target model.

of the source model. During the training, all parameters of the target model will be optimized. See model M_{T1} in Figure 2.3.

Before fine-tuning a model, one has to decide which weights of the source model should be copied. Intuitively, it makes sense to copy all weights up to a particular layer. A layer with random weights before layers with pretrained weights could undo the knowledge of the pretrained layer because it computes features that can not be handled by the next layer. The next layer will adapt.

As shown by Yosinski et al. [Yos+14], it makes sense to copy the weights up to one of the last layers of the model because the weights that need to adapt to the new task will still adapt and copying the weights improves generalization.

This option is more expensive because the whole deep learning model has to be trained. On the other hand, all layers of the model can adapt to the new dataset and task, which makes this option more powerful.

Combinations It is possible to combine the approaches to create a target model were some of the weights get copied and fixed, some of the weights get copied and fine-tuned and parts of the model change. See model M_{T3} in Figure 2.3 for an example.

Other methods for deep transfer learning have been proposed in the literature and will be discussed in Section 3.2.

Chapter 3

Related Work

This chapter lists the related work. Section 3.1 describes other work that is related to the segmentation of microscope images. In Section 3.2, work about transfer learning is analyzed. Finally, in Section 3.3 work to deploy deep learning methods is listed.

3.1 Segmentation

A segmentation assigns each pixel of an image a label. For cell segmentation, a common technique is first to obtain a foreground-background segmentation by thresholding the image values (e.g., [Ots79]) and then find connected foreground components. Meijering [Mei12] describes the standard techniques for cell segmentation. He finds that the standard approaches are intensity thresholding, feature detection, morphological filtering, region accumulation, and deformable model fitting. RACE [Ste+16], for example, is a framework for 3D cell segmentation that uses a hand-crafted segmentation pipeline.

Deep learning can be used to segment more complex images. Many methods have been proposed for semantic segmentation and instance segmentation of natural images. Hariharan et al. [Har+14] propose a method for simultaneous detection and segmentation that consists of four steps. They generate region proposals using MCG [Arb+14], use a CNN for feature extraction, classify the regions using an SVM, and finally refine the regions. Fully convolutional networks were introduced by Long and Shelhamer et al. [LSD15]. They proposed a model architecture for semantic segmentation that combines coarse, high layer information with fine, low layer information. This way, they can predict an accurate segmentation map. He et al. [He+17] introduced Mask R-CNN, which adds a branch to Faster R-CNN [Ren+15] that predicts an object mask to obtain an instance segmentation. They achieve outstanding results on the COCO [Lin+14] dataset and show that their method can also be used for other tasks like human pose estimation.

These methods can be used for segmentation of microscope images, but there are deep learning methods that have been introduced especially for segmenting these images. Ronneberger et al. [RFB15] introduced U-Nets for biomedical image segmentation. U-Nets have a novel architecture with long skip connections to recover fine details of the

image during the upsampling of the decoder network. They also introduce a weighted loss to separate touching objects and evaluate their architecture on two microscopy datasets. Most winning solutions of the 2018 Data Science Bowl use modified U-Nets[1]. Section 4.1 describes the U-Net architecture in more detail.

StarDist [Sch+18] is an advanced deep learning based method for instance segmentation that specializes in objects with roundish shapes. A StarDist model is based on a U-Net backbone but predicts star-shaped polygons that describe the objects. Therefore, StarDist is very powerful for images of crowded cells. They show that StarDist performs better than U-Nets and Mask R-CNN on a nuclei instance segmentation dataset. StarDist is the preferred method throughout the thesis, and Section 4.2 describes StarDist in more detail.

3.2 Transfer Learning

Transfer learning is a great approach to enable the training of powerful machine learning models on relatively small datasets.

Especially in computer vision, transfer learning has found a wide adaption. Many deep learning models are pretrained on ImageNet [Rus+15]. In 2014 Oquab et al. [Oqu+14] observed that image representations by CNNs that have been learned image classification on ImageNet could be used for object detection. They do not even fine-tune the convolutional layers trained on ImageNet but only add two fully connected layers to adapt. R-CNN [Gir+14] is a method that uses a CNN for feature extraction of proposed regions to do object detection. They pretrain the CNN on the ImageNet dataset. Redmon et al. [Red+16] presented a fast one-stage object detector called YOLO. They divide the image into a grid and predict bounding boxes and class probabilities for each grid cell. Their backbone network architecture is inspired by GoogLeNet [Sze+15] and pretrained on ImageNet. Lin et al. [Lin+17b] proposed a novel loss for object detection that focuses on hard examples during the training of a one-stage detector. As a backbone, they use a Feature Pyramid Network [Lin+17a] with a ResNet [He+16] encoder that was pretrained on ImageNet. The backbone used in Mask R-CNN (which has been described above) was also pretrained on ImageNet.

Despite the high usage of transfer learning in computer vision tasks, there is not much work on evaluating pretraining. He et al. [HGD18] discovered that ImageNet pretraining is not necessary to get state of the art results on the COCO dataset. They show that pretraining on ImageNet does not improve the final score on COCO if the model with random initialization is trained long enough. Yosinski et al. [Yos+14] had other results with an example based on ImageNet subsets. They had better classification scores for models that were trained to predict 500 classes of the ImageNet dataset if they were pretrained on the other 500 classes beforehand. On the other hand, they observed performance degradation if the tasks were less similar.

[1]First place: `https://www.kaggle.com/c/data-science-bowl-2018/discussion/54741`
Second place: `https://github.com/jacobkie/2018DSB`
Fourth place: `https://www.kaggle.com/c/data-science-bowl-2018/discussion/55118`

Other research on transfer learning was done by Kornblith et al. [KSL19] who explored if better ImageNet models do transfer better. They observe a strong positive correlation between the ImageNet accuracy and the transfer learning accuracy of models on multiple datasets. However, all their datasets contain only natural images.

In the processing of microscope images transfer learning has been used less. Falk et al. [Fal+18] provided U-Nets for transfer learning using an ImageJ Plugin, but they did not evaluate how the choice of the pretraining dataset affects the transfer learning performance. Oliveira et al. [OS18] proposed a transfer learning technique for chest radiographs that makes use of unsupervised image-to-image networks (like CycleGAN [Zhu+17], UNIT [LBK17], MUNIT [Hua+18]) to map images from the target domain to the source domain. The observed that their semi-supervised transfer worked better than fine-tuning on their chest radiographs.

For natural images, there is more work on how to improve transfer learning. Ngiam et al. [Ngi+18] weighted the loss during the pretraining on the model on specific samples of the source dataset such that the pretraining focuses on samples that are important for the target task. They find that the choice of pretraining data is more important than the size of the pretraining dataset. Therefore, their findings fit the conclusion of this thesis. Rozantsev et al. [RSF19] trained two models on the target and the source dataset simultaneously while forcing the weights of both models to be related. They introduce a weight regularizer that penalizes weights that are not a linear transformation of each other. This allows modeling a domain shift between the source and the target data.

3.3 Deep Learning Deployment

Fiji [Sch+12] is one of the most used tools from microscopic image analysis. The base of Fiji is ImageJ [SRE12] and ImageJ2 [Rue+17]. KNIME Image Processing [DB16] is built upon ImageJ2 and provides most of the functionality. Additionally, KNIME Analytics Platform [Ber+09] can be used to analyze the results of the analysis further.

Some efforts have been made to make deep learning based methods with microscope image analysis available in these tools. Weigert et al. [Wei+18] developed an ImageJ2 plugin to apply deep learning models in Fiji. They also released KNIME workflows to apply the same models. However, the training of the models needs to be done manually using a Python package. Falk et al. [Fal+18] provided an ImageJ plugin to apply and fine-tune U-Nets. Their tool is easy to use within Fiji and therefore integrates well into an image analysis pipeline for microscopy images. However, the plugin is not easily extensible to other methods (like StarDist) and does not provide the functionality to train models from scratch. The user has to fine-tune one of the provided models.

KNIME already provides a toolkit for deep learning[2] but using it to apply and train more complex deep learning models on microscope images is not straight forward. Therefore, this work adds another abstraction layer on top of this toolkit to make it easier to use.

[2] `https://www.knime.com/deeplearning`

Chapter 4

Method

In this chapter, the U-Net model architecture is described (Section 4.1). Based on this architecture, the StarDist model is explained (Section 4.2). Finally, a modification of the U-Net architecture based on residual shortcut connections is proposed, which improves the performance of StarDist models (Section 4.3).

4.1 U-Net

To obtain an instance segmentation, one can first compute a foreground-background segmentation and then find connected foreground components. If simple thresholding of the intensity values (e.g., Otsu [Ots79]) does not suffice, one can use more sophisticated methods to arrive at the foreground-background segmentation which will be used by the Connected-Component Labeling.

A U-Net is a deep convolutional neural network that was introduced for image segmentation [RFB15]. It is easy to train a U-Net to output 1 for foreground pixels and 0 for background pixels based on a ground truth segmentation. This U-Net can then be used to create a segmentation of the image which can be turned into an instance segmentation by a Connected-Component Labeling.

If the objects are crowded, the Connected-Component Labeling will assign the same label to different objects that are touching each other. In this case, a U-Net can be trained to predict three classes: foreground, background, and border (the boundary of one object) [Sch+18]. The border class will separate objects during the Connected-Component Labeling of the foreground class.

Alternatively, the objects can be separated with the background class and a weighted loss. Giving the pixel that are close to the border of multiple objects a higher loss during training will force the model to separate the objects [RFB15].

4.1.1 Architecture

A U-Net is built on the architecture of fully convolutional networks [LSD15]. The first part of the network successively reduces the spatial resolution of the image while increas-

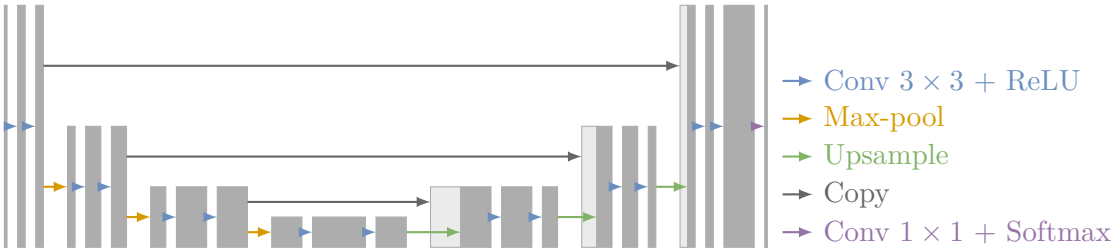


Figure 4.1: The architecture of a U-Net model. Each box corresponds to a three-dimensional tensor with spatial dimensions corresponding to the height of the box and the number of feature maps corresponding to the width of the box. Lightly filled boxes correspond to a copied tensor.

ing the number of feature maps (like backbones of many common architectures [KSH12; SZ15; He+16]).

To predict a segmentation map, the second part of the network increases the resolution again. This part differs from the architecture of fully convolutional networks because the upsampling is done successively (like the downsampling) using upsampling layers instead of pooling layers. To better localize features during the upsampling (the exact localization gets lost because of the pooling layers) feature maps from the downsampling part are concatenated with the upsampled feature maps (see Figure 4.1).

All operations of a U-Net require only a local input region. This means, that any U-Net can be applied to arbitrary image sizes (only limited by the GPU memory).

The architecture used in this work is slightly different from the architecture in the original paper. Padded convolutions instead of unpadded convolutions were used, and the 2×2 convolutional layer after the upsampling was omitted, but the number of feature maps was reduced during the second convolutional layer of each size.

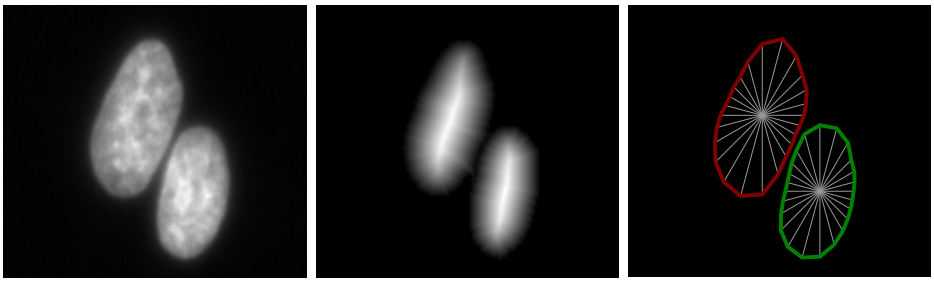
4.1.2 Training

A U-Net can be trained using training images and their corresponding ground truth segmentation. A pixel-wise soft-max computes the output probabilities, and the loss is a cross-entropy loss function per pixel.

The original paper [RFB15] used stochastic gradient descent with a batch size of only one image but high-resolution input images. A high momentum counteracted the training instabilities that could be caused by the small batch size. The training configuration used in this work is described in Section 5.3.

A weight map which increases the loss for pixels that are close to two or more objects was used by the authors of the U-Net paper [RFB15]. This forces the model to learn the separation border between objects.

In the KNIME Implementation the weight map was not used. Instead, a border class was used to separate the object in the 3-class U-Net model.



(a) Input (b) $d_{i,j}$ (c) $r^k_{i,j}$

Figure 4.2: Input and outputs of a StarDist model for an image of two cells.

4.2 StarDist

StarDist is a cell detection method that predicts a star-shaped polygon for each object describing the object's shape [Sch+18]. This polygon can be used directly to obtain an instance segmentation of the object. As we will see in Chapter 5 this instance segmentation is not perfect but can compete with other methods.

The foundation of the StarDist method is a convolutional neural network (CNN) that takes the normalized image as input and predicts a star-shaped polygon and a score for each pixel. On these polygon proposals, a score threshold and a non-maximum suppression (NMS) is applied to filter out false predictions.

4.2.1 Architecture

The StarDist model uses the network architecture of a U-Net (which is described in Section 4.1.1), but does not output a segmentation directly but the object scores $d_{i,j}$, and a map of polygon-rays $r^k_{i,j}$.

The $d_{i,j}$ output is trained to predict the normalized Euclidean distance to the background. This value is 0 for background pixels, 1 for the most centered pixel of an object, small for pixels close to the object boundary and large for pixels close to the object center. The $r^k_{i,j}$ output is trained to predict the length of rays from the pixel to the object boundary in n directions defining the polygon. Together the outputs yield one polygon proposal per pixel and its score (see Figure 4.2).

The polygon proposals are filtered by a score threshold and a non-maximum suppression to filter out false predictions and get only one polygon per object. The non-maximum suppression will prefer polygons whose center are close to the center of the object because of the trained score output (which is larger close to the center of an object). These polygons can capture the shape of the object better.

4.2.2 Training

During the training, a binary cross-entropy loss is optimized for the score outputs. The ray output is optimized using the mean absolute error weighted by the ground truth

score. This assures that no polygon has to be learned for background pixels where the score is zero. The model focuses on polygons close to the center of the object, which are favored by the non-maximum suppression.

4.2.3 Advantages

No merging of touching objects: Methods that first obtain a foreground-background segmentation and then find the instances by a Connected-Component Labeling are prone to merge cells that are close to each other. This is not an issue for StarDist because the model predicts shapes and instances in one go. The model is trained to detect centers of object instances and predict the distance to the object border, which is also valid for very crowded objects.

No suppression of crowded object: Methods that predict an object proposal by a bounding box like Mask R-CNN are prone to suppress objects that are close to each other and whose bounding boxes have a large overlap even if the objects itself do not overlap. StarDist does not have this issue because the NMS is applied to the shape proposals that represent the actual object shape much better than a bounding box and do not overlap much for crowded objects.

Relatively few parameters: Compared to Mask R-CNN StarDist has much fewer parameters and is, therefore, easier to train with less training data. StarDist has about 1.4 million parameters, while Mask R-CNN has about 45 million parameters.

4.3 Res-U-Net

The U-Net backbone limits the modeling capacity of a StarDist model. The U-Net backbone can be improved to increase the performance of a StarDist model. An intuitive way to improve the model capacity is by adding more layers to the model. However, adding more layers could also degrade the performance because it gets harder to optimize, as mentioned in Section 2.3.2. This decrease could be prevented by using residuals.

4.3.1 Architecture

For the Res-U-Net backbone, the two convolutional layers on each resolution level are replaced by residual blocks (see Figure 4.3). The first of the blocks always changes the number of feature maps to the desired number of feature maps for this level (left block in the figure). All subsequent blocks keep this number of feature maps (right block in the figure).

For the proposed architecture, two blocks are used at each resolution level (up and down) except for the second resolution level where three blocks are used. The numbers of feature maps for each resolution level is 64, 128, 256, and 512, respectively. Note that the 3×3 convolutional layers have only $1/4$ of the filters.

After each upsampling layer, a 2×2 convolution is applied, which halves the number of features maps. This is different from the used U-Net where no 2×2 convolution was

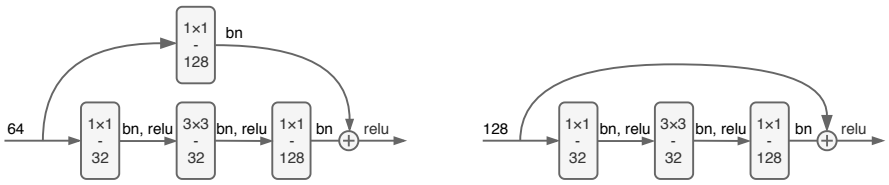


Figure 4.3: The two bottleneck building blocks of the Res-U-Net. The left block can be used with an arbitrary number of input features and can, therefore, be used to increase or decrease the number of features. The shown block increases the number of features from 64 to 128. The block on the right can only be used with a fixed number of input features and outputs the same number of features. The 3×3 convolution always has $1/4$ of the filters of the output convolution which reduces the number of parameters.

used, and where the last convolution reduced the number of feature maps at the previous resolution level.

The number of feature maps at each resolution level and the number of residual blocks at each resolution level can be changed easily to construct models with more or less modeling capacity. The described numbers were chosen because they improve the modeling capacity and performance of the model while keeping it relatively small. With the StarDist head applied to the proposed backbone, the model has slightly above 2 million trainable parameters.

Chapter 5

Experiments

In this chapter, the experiments to evaluate the applicability of transfer learning are described and discussed. First, the datasets (Section 5.1), evaluation metric (Section 5.2), and implementation details (Section 5.3) are described. Later, the motivation, description, and results of each experiment are listed and discussed (Section 5.10).

The general procedure for evaluating the transfer learning performance between a source dataset A and a target dataset B can be split into four parts:

1. Train a model on dataset A
2. Train a model on dataset B
3. Train a model on dataset B with 2, 5, 10, 50 and 200 training examples
4. Retrain the model of dataset A on dataset B with 2, 5, 10, 50 and 200 training examples

Most experiments use this procedure, and it will be called *transfer performance evaluation*.

The experiments will use the StarDist model. StarDist especially makes sense for instance segmentation on microscope images because it performs significantly better than usual U-Nets or Mask R-CNN [Sch+18]. The advantages of StarDist are described in Section 4.2.3.

5.1 Datasets

5.1.1 Simulating Microscopy

Obtaining ground-truth segmentations for real microscopic datasets is cumbersome if possible at all. Therefore it makes sense to simulate realistic microscope images with known ground truth segmentations to evaluate and pretrain models.

The tool *CytoGen*[1] was used to simulate images of HL60 cell nuclei and granulocyte nuclei.

[1] `https://cbia.fi.muni.cz/research/simulations/cytogen.html`

The simulation of microscope images of cell nuclei consists of the three main parts phantom generation, signal transmission, and signal detection and image formation [SKS09].

Phantom generation During the phantom generation, a model of the object is created. For microscopic simulations, many objects are too small to image them correctly (or observe them by eye) such that one can learn how the object looks like. Therefore, the model can only be built upon the expected appearance of the object.

Signal transmission The signal transmission describes how the signal of the perfect phantom is transmitted through an optical system to arrive at the imaging device. The primary degradation of the signal is blurring that can be described by the point spread function. Additionally, the light conditions might not be perfect (uneven illumination), which can cause different signal intensities in different regions of the image.

Image formation During the image formation, there are plenty of types of different noise that is introduced by the sensor and the electronic system.

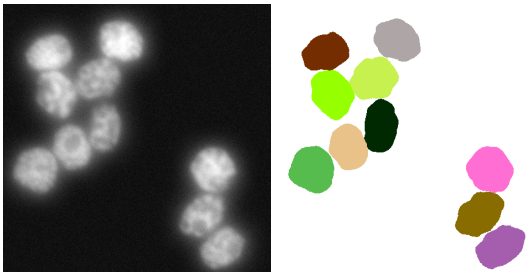
5.1.2 List of Datasets

Name	Category	Task	Image Size	#Images (train / test)
`HL60_LOW_NOISE`	F	Nuclei segmentation	484×484	850 / 150
`HL60_HIGH_NOISE`	F	Nuclei segmentation	484×484	850 / 150
`GRANULOCYTE`	F	Nuclei segmentation	484×484	850 / 150
`DSB2018`	F & B	Nuclei segmentation	varying	664 / 106
`STARDIST_DSB2018`	F	Nuclei segmentation	varying	447 / 50
`CITYSCAPES`	N	Segmentation	2048×1024	2975 / 500
`IMAGENET`	N	Classification	varying	1.2 M / n/a

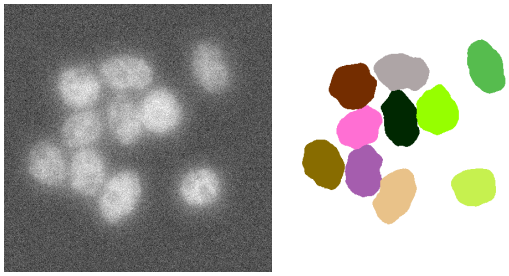
Table 5.1: List of experiment datasets. The category is denoted by F for fluorescence microscopy, B for bright-field microscopy, and N for natural images.

HL60_LOW_NOISE

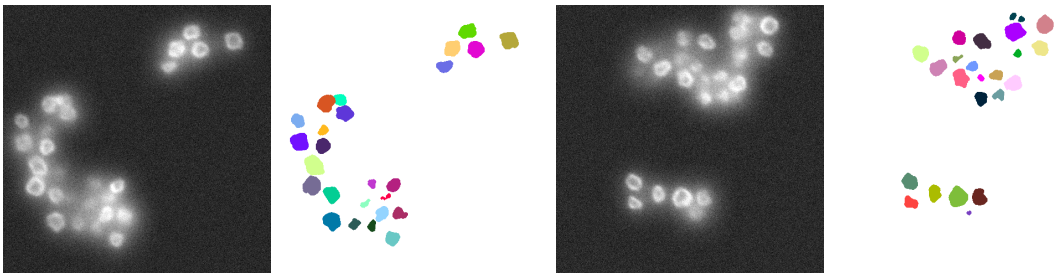
The `HL60_LOW_NOISE` dataset contains simulated fluorescence images of HL60 nuclei. The simulation was done with the tool *CytoGen*. The configuration was very similar to the example configuration released with the application. Changes were made in the acquisition time and dynamic range usage of the acquisition device such that the resulting images have low noise. Additionally, the image size was changed such that the



(a) HL60_LOW_NOISE



(b) HL60_HIGH_NOISE



(c) GRANULOCYTE

Figure 5.1: Example images and ground truth segmentations of the simulated datasets.

resulting images have a resolution of 484 × 484 and only one slice in the z-dimension. Each image contains ten nuclei phantoms that are positioned randomly. See Figure 5.1a for an example image.

HL60_HIGH_NOISE

The HL60_HIGH_NOISE dataset contains the same images as the HL60_LOW_NOISE dataset but with more noise. The higher signal to noise ratio was accomplished by reducing the acquisition time and dynamic range usage of the acquisition device. See Figure 5.1b for an example image.

GRANULOCYTE

The GRANULOCYTE dataset contains simulated fluorescence images of granulocyte nuclei. The simulation was done with the tool *CytoGen*. The configuration was very similar to the example configuration released with the application. The image size was changed such that the resulting images have a resolution of 484 × 484 and only one slice in the z-dimension.

A granulocyte nucleus consists of multiple thick elliptical parts called lobes which are connected by very thin channels that are not visible in the final images. The goal for this dataset is to segment each lobe individually. A KNIME workflow was used to disconnect the lobes in the ground truth segmentation and assign each lobe a unique identifier. See Figure 5.1c for example images.

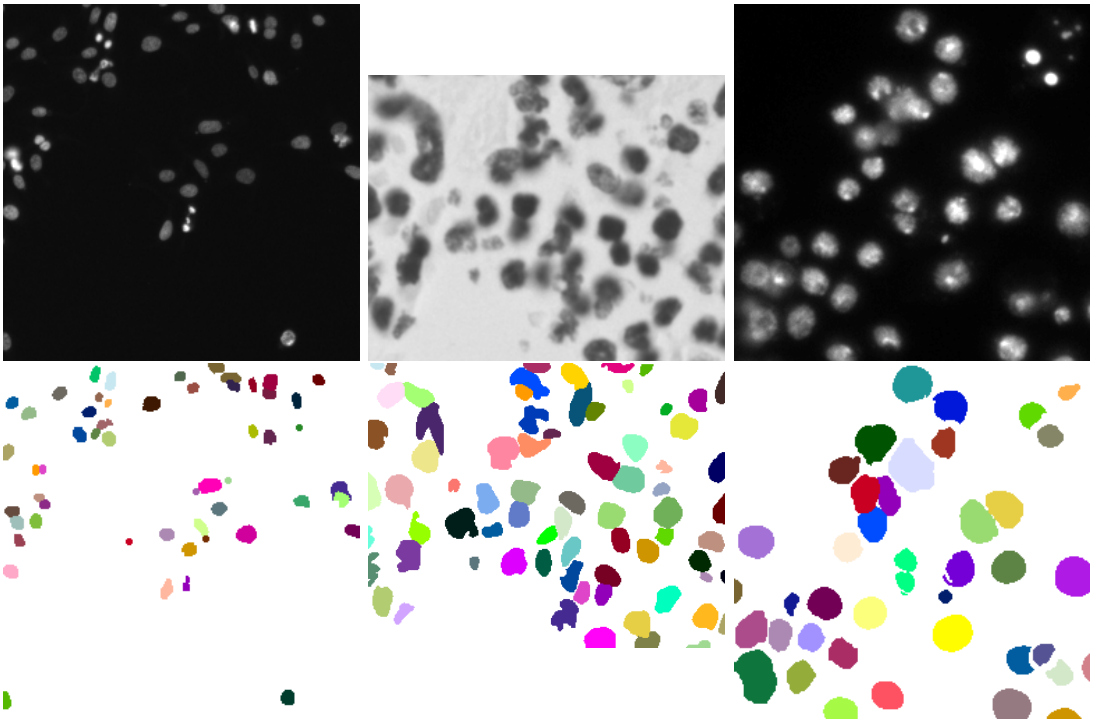


Figure 5.2: Example images and ground truth segmentations of the `DSB2018` dataset.

DSB2018

The `DSB2018` dataset contains real fluorescence and bright field images from the Kaggle competition *2018 Data Science Bowl*[2].

The competition website describes the dataset well.

> This dataset contains a large number of segmented nuclei images. The images were acquired under a variety of conditions and vary in the cell type, magnification, and imaging modality (brightfield vs. fluorescence). The dataset is designed to challenge an algorithm's ability to generalize across these variations. [Boo18]

Because of many labeling errors in the original dataset, a corrected dataset by Konstantin Lopuhin[3] was used.

See Figure 5.2 for example images.

STARDIST_DSB2018

The `STARDIST_DSB2018` dataset includes a subset of the `DSB2018` training images. Only images of fluorescence microscopy without obvious labeling errors are included. The

[2] `https://www.kaggle.com/c/data-science-bowl-2018/`

[3] `https://github.com/lopuhin/kaggle-dsbowl-2018-dataset-fixes/` commit `3035455`

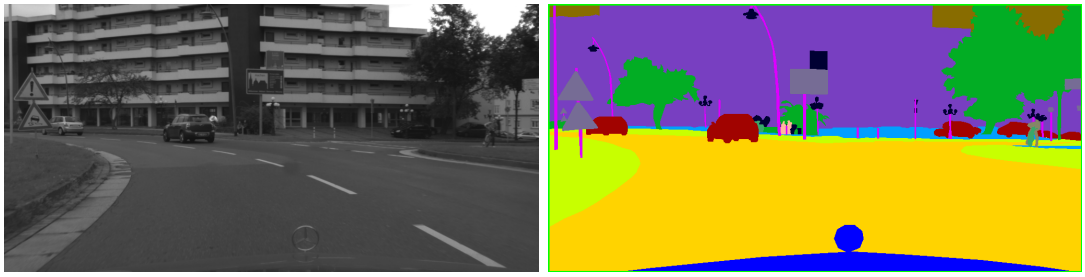


Figure 5.3: Example image and ground truth segmentation of the CITYSCAPES dataset.

dataset has been used in the StarDist paper [Sch+18] and can be downloaded from the GitHub repository of the project[4].

CITYSCAPES

The CITYSCAPES dataset contains images of urban street scenes of 50 cities in Germany. The images were taken at daytime throughout several months in spring, summer, and fall at good to medium weather conditions. Dense instance-wise ground truth segmentations for 30 classes are provided [Cor+16].

This dataset has been chosen to evaluate if pretraining on real-world images has a valuable effect on the performance of models for microscope images. The dense ground truth segmentations allows training of full encoder-decoder models. The models can be reused easily for a nuclei segmentation task on microscope images. See Figure 5.3 for an example image.

IMAGENET

The IMAGENET dataset contains many real-world photographs. During the experiments, no model was trained on this large dataset, but a model is used that has been trained on the ImageNet 2012 classification dataset [Rus+15].

5.2 Evaluation Metric

For the experiment evaluation, the mean Average Precision (mAP) was used as defined for the 2018 Data Science Bowl[5]. Before we define the mAP, we define some requirements.

Definition 5.1 (Segment)**.** *A segment $A \subset \Omega$ of a segmentation s is the set of all points that are assigned to label $l \in L$.* segments(s) *is the set of all segments of the segmentation s.*

[4] https://github.com/mpicbg-csbd/stardist/releases/tag/0.1.0
[5] https://www.kaggle.com/c/data-science-bowl-2018/overview/evaluation

Definition 5.2 (Intersection over Union (IoU))**.** *For segments $A \subset \Omega$ and $B \subset \Omega$, the Intersection over Union (also Jaccard index) is defined as:*

$$\mathrm{IoU}(A, B) := \frac{|A \cap B|}{|A \cup B|}$$

Definition 5.3 (True Positives, False Positives, False Negatives)**.** *For an instance segmentation $s : \Omega \to L$, a predicted instance segmentation $\hat{s} : \Omega \to L$, and a threshold value $0.5 \leq t \leq 1$ we define we define the number of true positives (TP), false positives (FP) and false negatives (FN) as follows:*

$$\begin{aligned}
\mathrm{TP}_t(s, \hat{s}) &:= |C| \\
\mathrm{FP}_t(s, \hat{s}) &:= |\{A \in \mathrm{segments}(\hat{s}) \mid A \notin C\}| \\
\mathrm{FN}_t(s, \hat{s}) &:= |\{B \in \mathrm{segments}(s) \mid \neg \exists A \in C : \mathrm{IoU}(A, B) > t\}|
\end{aligned}$$

With C being the set of correctly predicted segments:

$$C := \{A \in \mathrm{segments}(\hat{s}) \mid \exists B \in \mathrm{segments}(s) : \mathrm{IoU}(A, B) > t\} \tag{5.1}$$

Using the notion of True Positives, False Positives, and False Negatives, we can finally define the mean Average Precision.

Definition 5.4 (Mean Average Precision)**.** *For an instance segmentation $s : \Omega \to L$ and a predicted instance segmentation $\hat{s} : \Omega \to L$, the mean Average Precision is defined as follows:*

$$\mathrm{mAP}(s, \hat{s}) = \frac{1}{|T|} \sum_{t \in T} \frac{\mathrm{TP}_t(s, \hat{s})}{\mathrm{TP}_t(s, \hat{s}) + \mathrm{FP}_t(s, \hat{s}) + \mathrm{FN}_t(s, \hat{s})} \tag{5.2}$$

With the IoU thresholds $T = \{0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95\}$.

The mean Average Precision of a whole dataset is the mean over the mAP *for each segmentation.*

Note that there are other definitions of the Average Precision that usually measure the area under the precision-recall curve. In this work, the more straightforward metric from the 2018 Data Science Bowl will be used because it requires no score value for segments and results are comparable with the results of the 2018 Data Science Bowl.

5.3 Implementation Details

The experiments were implemented in Keras [Cho+15] with the TensorFlow [Aba+16] backend. The code can be found at
`https://github.com/HedgehogCode/keras-transfer-learning`.

Augmenter	Parameters
Affine	scale: {HL60_LOW_NOISE : $[0.2, 1.2]$, GRANULOCYTE : $[0.5, 1.7]$}, rotation: $[0, 360]$, shear: $[-16, 16]$
CropToFixedSize	width: 256, height: 256
Fliplr	p: 0.5
Flipud	p: 0.5
GammaContrast	gamma: $[0.5, 2.0]$
Sharpen	alpha: $[0.0, 1.0]$
Emboss	alpha: $[0.0, 1.0]$, strength: $[0.5, 1.5]$
Add	value: $[-0.1, 0.1]$
AdditiveGaussianNoise	scale: $[0, 0.2]$
GaussianBlur	sigma: $[0, 3]$
MotionBlur	
Invert	p: 0.3

Table 5.2: List of data augmentation methods of the *imgaug* library applied to the HL60_LOW_NOISE and GRANULOCYTE dataset for the *Combining Simulated Datasets* experiment.

Dataset Splits Every dataset was split into a training and testing set. Furthermore, 10 percent of the training images were excluded from the training and used as a validation set (2 percent for the CITYSCAPES dataset). If the number of training examples was reduced, this did not affect on the validation dataset, which was still the same as for the full training dataset.

Training The training was done with a batch size of 8 and an image size of 256×256. The patches were cropped randomly from the full-sized image and randomly flipped vertically and horizontally. After 512 patches or 64 batches (1 step), the loss was computed on the full-sized images of the validation set.

The Adam optimizer [KB15] was used for all models with a learning rate of 0.001, which was reduced by a factor of 5 if the validation loss did not improve for eight steps. The training was stopped once the validation loss did not improve for 12 steps.

The model with the ResNet50 [He+16] backbone (for the experiment in Section 5.6) was trained with a batch size of 2; the learning rate was reduced if the validation loss did not improve for 30 steps and the training was stopped if the validation loss did not improve for 40 steps.

Data Augmentation The data augmentation was done with the *imgaug* library[6]. For the *Combining Simulated Datasets* experiment (Section 5.9) multiple data augmentation techniques were applied. The augmenters are listed in Table 5.2.

[6] https://github.com/aleju/imgaug

5.4 Experiment: StarDist with Res-U-Net Backbone

Motivation StarDist is a handy tool for instance segmentation of roundish shapes, but the accuracy could be improved if a more powerful backbone was used. The StarDist model in the original paper uses a usual U-Net backbone, but U-Net backbones have been improved successfully (For example, for the Kaggle 2018 Data Science Bowl competition).

Description The usual U-Net was replaced by a Res-U-Net which is described in Section 4.3. The StarDist model with the Res-U-Net backbone was compared to the StarDist model with the usual U-Net backbone on the `STARDIST_DSB2018` dataset (to compare the results to the StarDist paper) and on the `DSB2018` dataset to highlight the improved modeling capacity. Additionally, a *Plain-U-Net* backbone was tested by using the same architecture as the Res-U-Net but without the short residual skip connections. This backbone was added to ensure that the performance gain does result from the residual architecture and not just from adding more layers.

Results Table 5.3 shows the Average Precision of the StarDist model with Res-U-Net backbone on the `STARDIST_DSB2018` dataset. We observe that the performance is better than the performance of StarDist that was reported by Schmidt and Weigert et al. [Sch+18]. The Plain-U-Net performs much worse than the Res-U-Net and even worse than the StarDist model for most IoU thresholds.

The mean Average Precision of StarDist models with both backbones is shown in Figure 5.4. On the `STARDIST_DSB2018` dataset the performance of the models only differs slightly, but on the `DSB2018` dataset the performance of the Res-U-Net backbone is much better. The Plain-U-Net performs worse than both other backbones on both datasets.

	Threshold	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90
[Sch+18]	U-Net (2 class)	.674	.630	.598	.565	.534	.482	.415	.325	.203
	U-Net (3 class)	.806	.775	.743	.701	.654	.578	.491	.374	**.226**
	Mask R-CNN	.832	.805	.773	.730	.684	.598	.489	.353	.189
	StarDist	.864	.836	.804	.755	.685	.586	.450	.287	.119
	StarDist (Res-U-Net)	**.871**	**.849**	**.823**	**.783**	**.725**	**.640**	**.529**	**.384**	.192
	StarDist (Plain-U-Net)	.853	.831	.797	.749	.680	.583	.460	.294	.111

Table 5.3: Average Precision of the StarDist model with a Res-U-Net backbone at different IoU threshold values compared to the results from the StarDist paper [Sch+18]. The reported value is the mean value of six models.

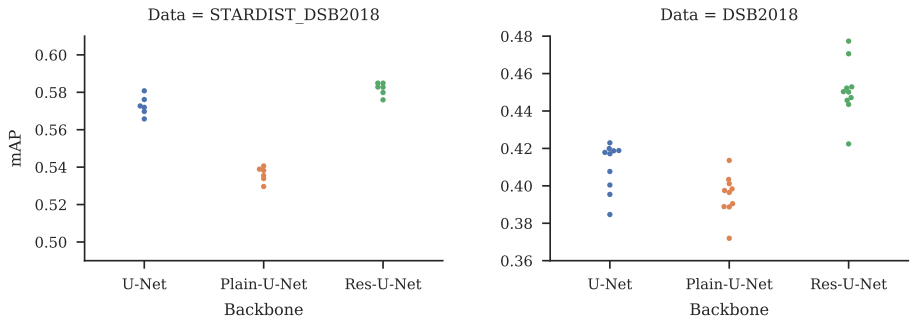


Figure 5.4: Scores of a StarDist model with the usual U-Net backbone, a Plain-U-Net (like Res-U-Net but without the shortcut connections) and with the improved Res-U-Net backbone for the `STARDIST_DSB2018` and `DSB2018` datasets.

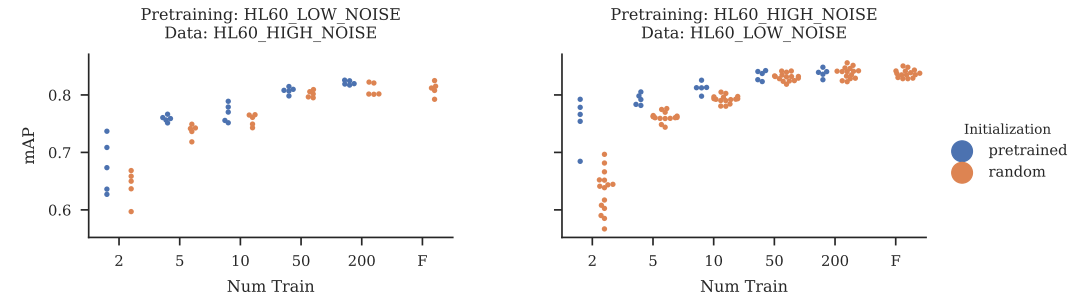
5.5 Experiment: Different Noise Levels

Motivation During a biological experiment, the configuration of the image acquisition can change. This can be due to new requirements that come up during the experiment (e.g., lower light power to protect the sample) or due to new opportunities (e.g., new microscope).

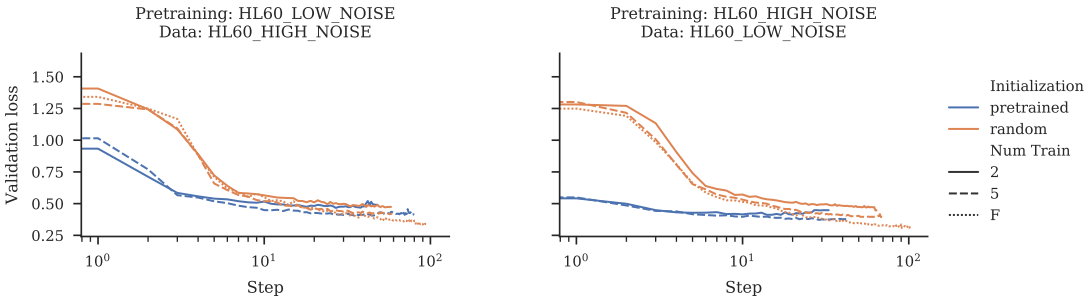
A concrete example would be that the acquisition speed needs would need to be improved because tracking of cells was not possible in the previously recorded time series. To reduce the acquisition time to achieve higher frame rates, one can reduce the spatial resolution or light exposure. If the resolution should not be reduced anymore because of the size of the imaged object, reducing light exposure is the only option. This will result in images that have a lower signal to noise ratio than the previous images. A lot of manual annotation labor can be saved if the model from the previous analysis (on the less noisy images) could be reused.

Description The `HL60_LOW_NOISE` and `HL60_HIGH_NOISE` dataset simulate the explained example. The `HL60_HIGH_NOISE` dataset has a lower signal to noise ratio because of the reduced simulated acquisition time and dynamic range usage. A StarDist model with a usual U-Net backbone was used to evaluate the transfer performance in both directions.

Results The results of the transfer performance evaluation are shown in Figure 5.5a. We can see that the pretrained models mostly perform better than the models with random initialization for a low number of training examples. For only two training examples, all models (with pretrained and random initialization) vary strongly in their final performance. This effect vanishes with the usage of five training example. The gap between models with pretrained and random initialization starts to close with ten training examples. This is the case because the dataset is rather simple and very uniform.



(a) Final evaluation scores.



(b) Validation loss history during training.

Figure 5.5: Final scores and validation loss history of models that have been pretrained on datasets with a different noise level.

The improvement is more significant if the model was pretrained on the dataset with high noise.

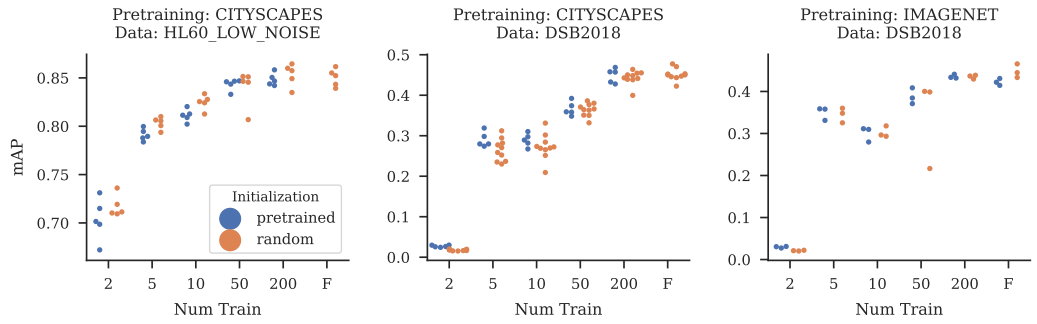
Figure 5.5b shows the validation loss history of the models during the training. We can see that models that have been pretrained learn much quicker.

5.6 Experiment: Pretraining on Natural Images

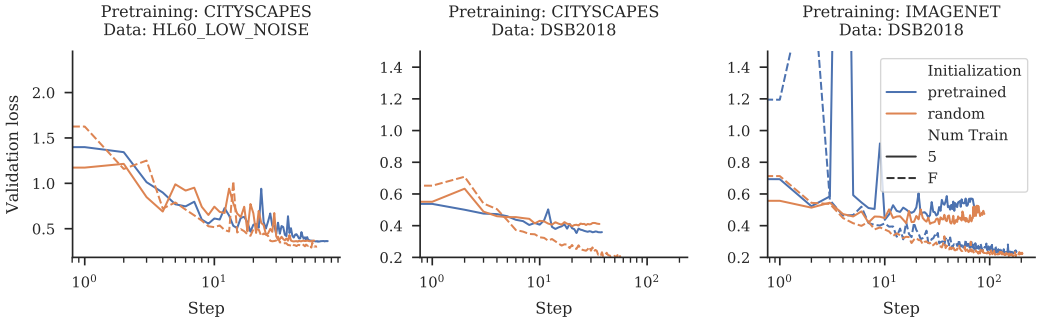
Motivation It has become common to use ImageNet pretraining in computer vision tasks. The ImageNet dataset is used because it is huge and diverse and the learned features proofed to be useful for many datasets of natural images.

Images of microscopes are not comparable with natural images, but the features that were learned for natural images might still be useful for them. Therefore, it would be interesting to find out if pretraining on large datasets of natural images helps models for microscopic image analysis.

Description The `CITYSCAPES` dataset was used to pretrain models because it is a relatively large dataset of natural images with segmentation ground truth. The segmentation ground truth allows training models with the same backbone architecture



(a) Final evaluation scores.



(b) Validation loss history during training.

Figure 5.6: Final scores and validation loss history of models that have been pretrained on datasets of natural images.

(Res-U-Net) as needed for the StarDist models on microscope images. The pretrained models were used on the `DSB2018` dataset and the `HL60_LOW_NOISE` dataset.

Additionally, a backbone that was trained on the `IMAGENET` dataset was used. The `IMAGENET` dataset does not provide ground truth segmentation. Therefore, the architecture of the backbone needed to be extended in order to make it applicable. A pretrained ResNet50 [He+16] was used for the encoder part of the model while the decoder part was not pretrained.

Results Figure 5.6a shows the transfer performance evaluation. The pretraining on `CITYSCAPES` seems to have a slight negative influence for the `HL60_LOW_NOISE` dataset. The best model for each number of training examples is always a model with random initialization. On the other hand, for the `DSB2018` dataset, the pretraining on the `CITYSCAPES` dataset does not seem to have a negative influence. However, it also does not seem to have a positive influence (Except for the experiment with five training examples). Also using a model pretrained on the `IMAGENET` dataset does not improve the final scores.

The validation loss history of all models is shown in Figure 5.6b. We observe similar

learning curves for the pretrained models as for the randomly initialized models. Hence, pretraining on natural images does not help to train models faster.

5.7 Experiment: Pretraining on Simulated Data

Motivation Obtaining ground-truth segmentations for a dataset is cumbersome and labor-intensive. For real-world computer vision tasks, there are large labeled datasets that can be used for pretraining. For microscopic image analysis, the images can differ vastly, and there could be no available dataset that is large and similar enough to use it for pretraining. In such cases, it could be helpful to simulate training images with known ground truth labels. These labels can be used to pretrain a model which will adapt quickly to the target dataset.

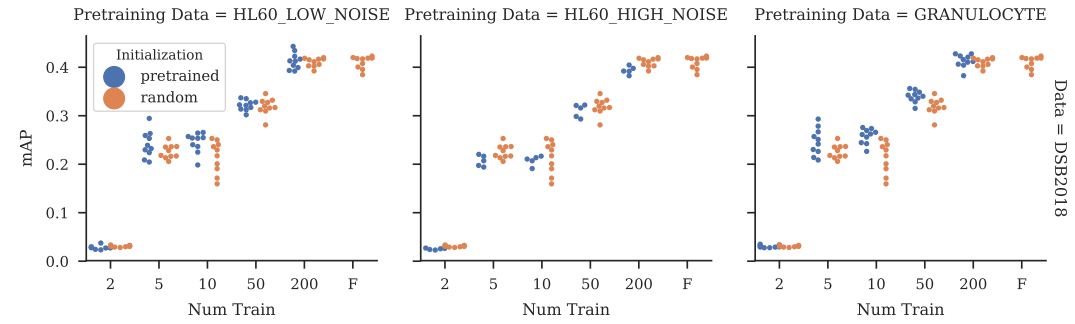
Description The transfer performance was for models that were pretrained on each simulated datasets (`HL60_LOW_NOISE`, `HL60_HIGH_NOISE`, `GRANULOCYTE`) and later trained for the `DSB2018` dataset. The StarDist model with the usual U-Net backbone was used.

Results The final scores for the transfer performance evaluation are shown in Figure 5.7a. We observe that pretraining on the `HL60_LOW_NOISE` dataset only improves the performance slightly if at all and pretraining on the `HL60_HIGH_NOISE` dataset does not improve the performance. This could be due to the simple nature of the HL60 based datasets and the model trained on the `HL60_HIGH_NOISE` dataset probably focuses on learning how to handle the noise, which is less critical on the `DSB2018` dataset. The improvements are greater for models that have been pretrained on the `GRANULOCYTE` dataset. This makes sense because the `GRANULOCYTE` dataset is more challenging, and the model has to learn how to handle nuclei of very different sizes and nuclei that are not in focus.

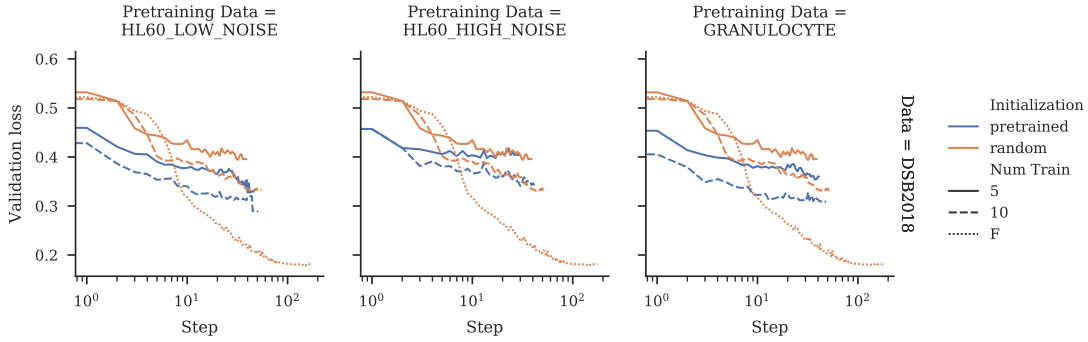
Figure 5.7b shows the validation loss history of all models. We observe that the loss starts much lower for the pretrained models and therefore they achieve better values faster.

5.8 Experiment: Pretraining on DSB2018

Motivation The `DSB2018` dataset contains images of fluorescence and bright-field microscopy which have been acquired under many different conditions. Therefore, the dataset is very diverse in the field of light microscope images (like ImageNet for natural images). Additionally, the dataset is large compared to other labeled datasets of microscope images. Since ImageNet proved to be a good candidate for the pretraining of models on natural images, the `DSB2018` dataset might be a promising candidate for the pretraining models on microscope images.



(a) Final evaluation scores.



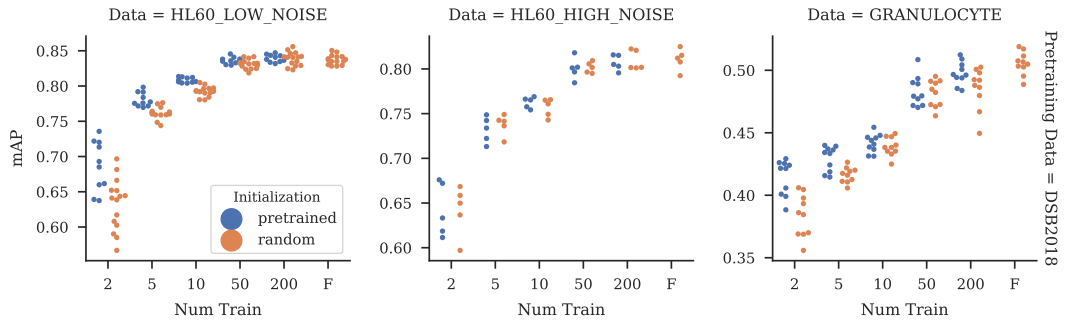
(b) Validation loss history during training.

Figure 5.7: Final scores and validation loss history of models on `DSB2018` that have been pretrained on simulated datasets.

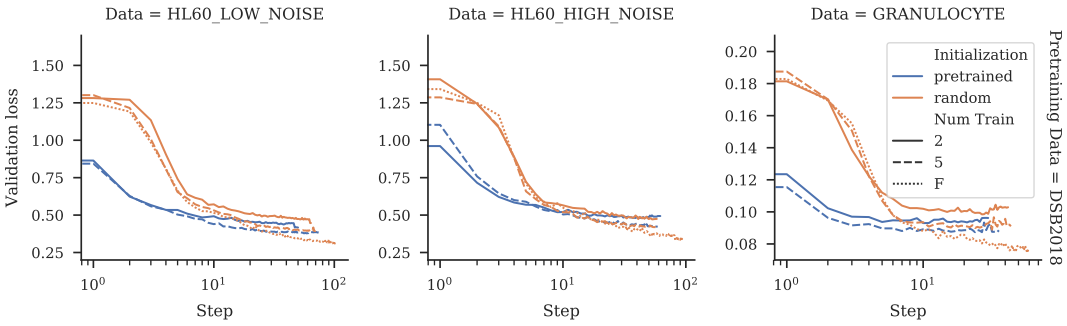
Description A transfer performance evaluation was performed using the `DSB2018` dataset as the pretraining dataset and each simulated dataset (`HL60_LOW_NOISE`, `HL60_HIGH_NOISE`, `GRANULOCYTE`) as the target dataset. A StarDist model with the usual U-Net backbone was used.

Results Figure 5.8a shows the final scores of the transfer performance evaluation. We can see that the pretraining helps to improve the scores significantly for the `HL60_LOW_NOISE` and the `GRANULOCYTE` dataset if only few training images are used. For the `HL60_HIGH_NOISE` dataset the pretraining does not help for the final scores. This could be due to the low signal to noise ratio that cannot be handled by the model pretrained on `DSB2018`.

The validation loss history, which is shown in Figure 5.8b shows that for all datasets, the pretrained models fit much quicker.



(a) Final evaluation scores.



(b) Validation loss history during training.

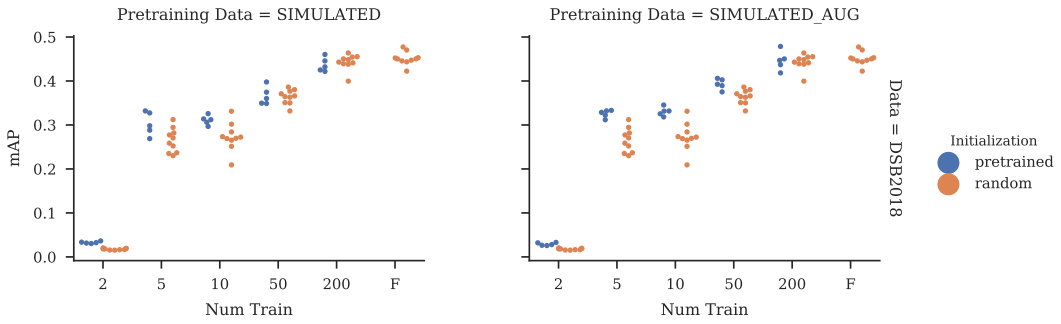
Figure 5.8: Final scores and validation loss history of models that have been pretrained on the `DSB2018` dataset.

5.9 Experiment: Combining Simulated Datasets for Pretraining

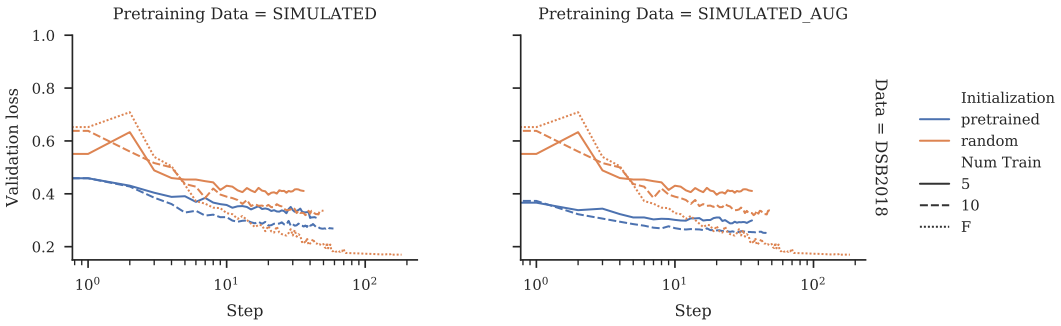
Motivation Combining the motivations of the previous two experiments leads to the conclusion that a diverse and challenging dataset of simulated images would be suited well for pretraining. It is possible to simulate enormous datasets, which can be made more diverse by using different simulators and different simulator settings. Additionally, the diversity of the data can be improved by data augmentation. For example, inverting the intensities of fluorescence images can make them look more like bright-field images.

Description For this experiment, the model was pretrained not only on one dataset but on both the `HL60_LOW_NOISE` and the `GRANULOCYTE` dataset. During the training of the backbone, the dataset was changed every 15 steps, and the backbone weights were reused. The model was pretrained for a total of 300 steps.

The same experiment was repeated with data augmentation to increase the diversity of the pretraining dataset (Section 5.3 contains the list of data augmentation techniques).



(a) Final evaluation scores.



(b) Validation loss history during training.

Figure 5.9: Final scores and validation loss history of models that have been pretrained on multiple simulated datasets.

Results As shown in Figure 5.9a, the scores of the models improved significantly if the weight were pretrained. We can see that data augmentation helps to improve the scores even further.

Figure 5.9b shows the validation loss history of the models. We can see that the validation loss starts much lower for pretrained models—especially if the model was pretrained with data augmentation.

5.10 Discussion

The experiments showed that transfer learning of the StarDist model can improve the performance on small datasets of microscope images. We observe that the pretraining dataset has to be related to the new dataset. As shown in Section 5.5, if the datasets only differ in noise levels, the model can more easily adapt to the new data (especially if the new data is easier than the pretraining data).

If the datasets are not that closely related, the source dataset has to contain images that are visually similar to the images of the target dataset. We observe this in multiple experiments. In Section 5.6, the models are pretrained on natural images that are not

related and not similar to the target dataset. The performance of the models does not benefit from this pretraining. In Section 5.7, the models are pretrained on simulated images that are very uniform and therefore only related to some of the images of the `DSB2018` dataset. For example, none of the simulated datasets contains images that look like they are from bright field microscopy, but the `DSB2018` dataset includes bright field images. The benefit for the pretrained images is only small if there is any. The `DSB2018` is more diverse and contains images that are related to any of the simulated datasets. In Section 5.8, we can see that `DSB2018` pretraining improves the models on these datasets. In the last experiment in Section 5.9, the simulated datasets are combined and augmented such that the pretraining is more diverse and closer related to the target images. We observe that the model benefits from this pretraining much more than just using one of the simulated datasets on its own.

A significant observation is that an appropriate pretraining can reduce the training time significantly. Pretrained models start at a much lower validation loss and drop quicker to values that are very close to the final validation loss. This can be applied in practice because the models can be trained on lower-tier hardware within a reasonable time. Also, the analyst does not have to wait that long until he gets reasonable results that can be used to analyze the data further.

Chapter 6

KNIME Implementation

In this chapter, a framework for instance segmentation in KNIME Analytics Platform is introduced. KNIME Analytics Platform [Ber+09] is a software that allows building data analysis workflows with a graphical programming interface. The workflows consist of connected processing steps that are implemented by KNIME Nodes.

Microscopes can produce large images that need to be processed automatically. Therefore, the framework is required to process large images. However, processing a large image is often not that easy because the images itself or the processing techniques can require more memory than available on the machine. In this case, one has to come up with processing techniques that do not require the whole image at once. The KNIME Tensor Processing framework addresses these issues by providing means to define a computation graph that can be executed in a tiled fashion and therefore, never requires the whole image to be loaded into memory. The Tensor Processing framework is described in more detail in Section 6.1.

Another requirement, which is directly derived from the desired users of the developed framework (e.g., biologists), is high usability without the need for programming. Ideally, the framework uses software and usage patterns that are already known to the user. KNIME Analytics Platform is easy to use via a visual programming interface that enables non-programmers to create complex data analysis pipelines. It is already used by many biologists and provides many tools to analyze the data further after the instance segmentation. Therefore the KNIME Instance Segmentation framework, which is described in Section 6.2, was developed.

6.1 Tensor Processing Framework

The KNIME Tensor Processing framework enables programmers to build a computation graph by defining single operations on tensors and stacking them together. This computation graph can then be executed on different input data. The graph executor takes care of executing the operations only on tiles of the image, caching intermediate results, and scheduling the execution order. The programmer only has to take care that the defined operations can be executed in a tiled fashion. See Figure 6.1 for an illustration

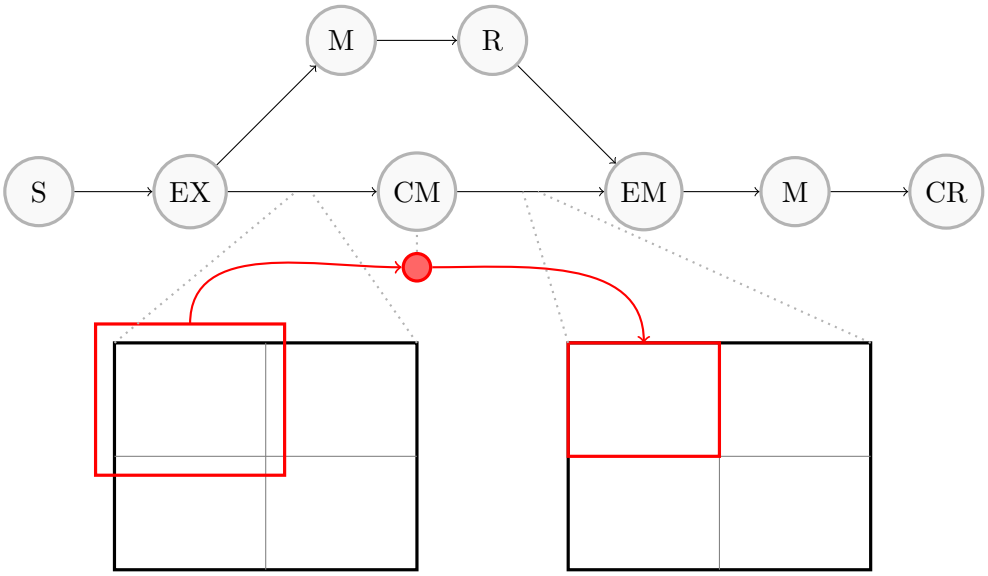


Figure 6.1: Illustration of the idea of the KNIME Tensor Processing framework. The user defines a graph of operations which is later executed. The boxes represent images before and after a context map (CM) operation. The context map operates only on a tile of the image but needs a large tile of the input image (drawn in red).

of a computation graph and a context map operation (which executes an operation on a tile of a tensor using some additional context).

KNIME Implementation

In KNIME Analytics Platform, the Tensor Processing framework is used in cells (entries in a KNIME table) that contain either an evaluated tensor or a computation graph and references on the input tensors. KNIME Nodes can add nodes to this computation graph and either execute it directly or save the extended computation graph in a new cell. This extending of the computation graph results in lazy evaluation that allows the KNIME Nodes to be executed quickly and prevents the saving of uninteresting intermediate results which is handy for data augmentation and the processing of large images.

Optimization Opportunities

The formulation of the processing pipeline as a computation graph allows for many potential optimizations. The static graph can be optimized by merging equal nodes (to prevent running an operation multiple times) or moving crops (to prevent running operations on data that is cropped away). The execution of the graph can be optimized by smart caching of intermediate results (if the caching overhead is lower than the cost of executing the operation again) and an execution order that makes maximum use of the available resources.

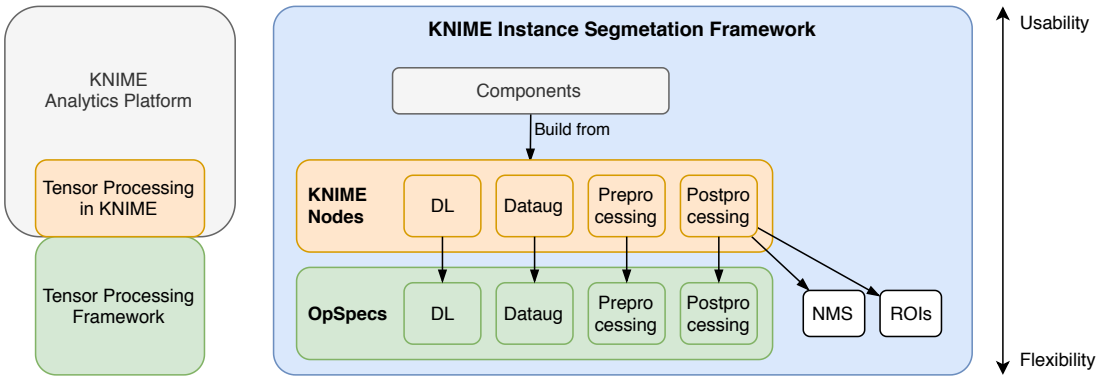


Figure 6.2: Parts of the KNIME Instance Segmentation framework.

The general formulation of the graph allows switching the execution strategy to something more efficient quickly.

6.2 KNIME Instance Segmentation Framework

The Instance Segmentation framework was created to enable the usage of instance segmentation models in KNIME Analytics Platform. At the highest level, the framework makes use of shared Components (A Component is a KNIME Node that contains a workflow consisting of multiple KNIME Nodes itself). These Components can be reused in other workflows to train and apply instance segmentation models. They are documented implicitly by the encapsulate workflow which can be changed to change their behavior.

The shared Components are build from KNIME Nodes that are either provided by KNIME Analytics Platform and its official extensions, the Tensor Processing framework, or the KNIME Instance Segmentation framework. The KNIME Nodes of the KNIME Instance Segmentation framework provide mainly functionality to execute deep learning models and apply data augmentation using the computation graph. Also, they provide preprocessing and post-processing procedures needed by the implemented models. The KNIME Nodes do not encapsulate the logic and algorithms but rely on lower level implementations that can be reused.

See Figure 6.2 for an overview of the main parts of the KNIME Instance Segmentation framework.

The framework can be extended by creating new Components that rely on existing KNIME Nodes, implementing new KNIME Nodes that rely on existing operations and algorithms, or implementing new operations and algorithms.

6.2.1 Shared Components

***Load/Generate/Read [. . .]*:** Load/Generate/Read the datasets used in the Example Workflows section. Each Node has two table outputs, one for the training data

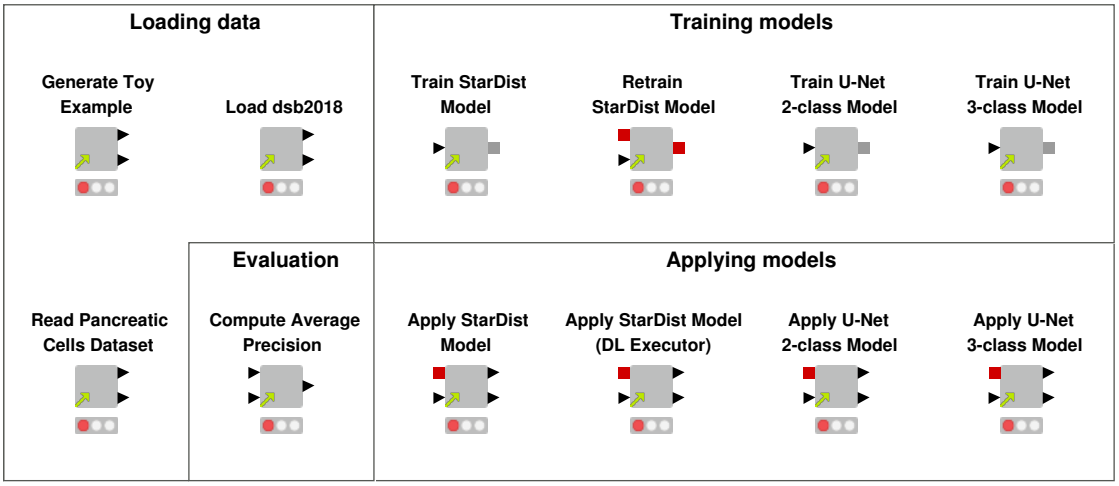


Figure 6.3: Shared components of the framework.

and one for the testing data. The Nodes can be swapped.

***Train StarDist Model*:** Creates and trains a StarDist model. The data augmentation and preparation is done by adding operations to the computation graph. The training is done by calling the Keras API [Cho+15] from the KNIME Python Integration. See Figure A.5a for a simplified version of the encapsulated workflow.

***Retrain StarDist Model*:** Fine-tunes the given StarDist model on the given training data. The Node is very similar to the *Train StarDist Model* Node. However, it does not create the StarDist model but uses the backbone of the input model.

***Train U-Net [2,3]-class Model*:** Creates and trains U-Net models. The input and output ports correspond to the ports of the *Train StarDist Model* Node and all training from scratch Nodes can be swapped.

***Apply StarDist Model*:** Applies a StarDist model to input images. The execution of the deep learning model is done by an operation that is added to the computation graph. The backbone of the StarDist model is a U-Net [RFB15], and the whole model contains only convolutional layers, upsampling layers, and max-pooling layers which makes it applicable to images of arbitrary size. It can also be executed on tiles of an image if a sufficient context is provided to prevent border artifacts. The *Tensor TensorFlow Executor* Node implements this tiled execution of the model. After the deep learning model execution, the non-maximum suppression is executed on the whole image at once. The Component outputs a segmentation for each input image and the individual segments along with their score. See Figure A.5b for a simplified version of the encapsulated workflow.

***Apply StarDist Model (DL Executor)*:** Applies a StarDist model without the use of the *Tensor TensorFlow Executor* Node but with the *DL Network Executor* (from

the KNIME Deep Learning Integration) which executes the model on the whole image at once. In Section 6.3.4, the limitations of this implementation will be shown.

***Apply U-Net [2,3]-class Model*:** Applies a U-Net model on the input images. The input and output ports correspond to the ports of the *Apply StarDist Model* Node, and all apply-nodes can be swapped.

***Compute Average Precision*:** Compares predicted segments to a ground truth segmentation and computes the Average Precision of the prediction for multiple intersection over union (IoU) thresholds.

See Figure 6.3 for all developed Components.

6.3 Example Workflows

KNIME Workflows that train and evaluate a StarDist model, a 2-class U-Net and a 3-class U-Net were built using the developed framework.

The first two examples demonstrate how the framework can be used to train all of the mentioned models from scratch on new data. Section 6.3.1 demonstrates and evaluates the models on an easy artificial dataset. In Section 6.3.2, the models are used on the `STARDIST_DSB2018` dataset. Transfer learning of a StarDist model is demonstrated in Section 6.3.3 using a dataset with a very small amount of labeled training data. Finally, the framework is used to label a huge image in Section 6.3.4.

Screenshots of the developed KNIME workflows are shown in Appendix A.

6.3.1 Toy Example (TOY)

The `TOY` dataset consists of images of touching half-ellipses. The dataset was specifically designed to showcase the limitations of methods that do an NMS on bounding boxes of the object instances (like Mask R-CNN). The bounding boxes of the half-ellipses can overlap significantly, which can cause one of the ellipses to be suppressed by the NMS. Mask R-CNN is not yet implemented in the framework, but we will still use the dataset because it is an accessible dataset to test and demonstrate the models. The dataset consists of a total of 500 images (475 training, 25 testing) with dimensions of 256×256. Some example images from the dataset are shown in Figure 6.4.

Three workflows were built (one workflow for each model) to evaluate the models on the `TOY` dataset using the developed framework. The workflows generate the training and testing examples (the generator uses a fixed seed), train the model on the training data, apply the model on the testing data and compute the Average Precision using the ground truth segmentation of the testing data. The workflows only differ in the *Train <model-name>* and *Apply <model-name>* Nodes. A screenshot of the workflow for the 3-class U-Net is shown in Figure A.1.

Table 6.1 shows the Average Precision scores for each model. As expected, StarDist is superior to the U-Net models and obtains perfect detections for lower IoU thresholds.

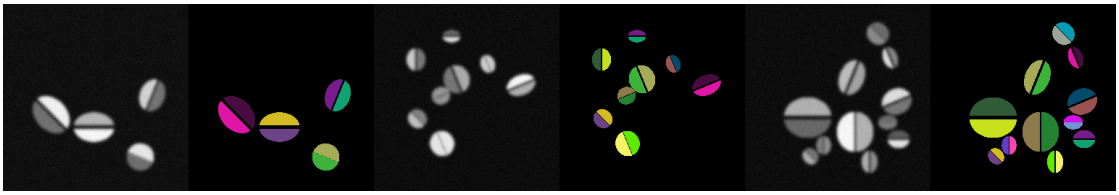


Figure 6.4: Some images from the generated `TOY` dataset along with their ground truth instance segmentations.

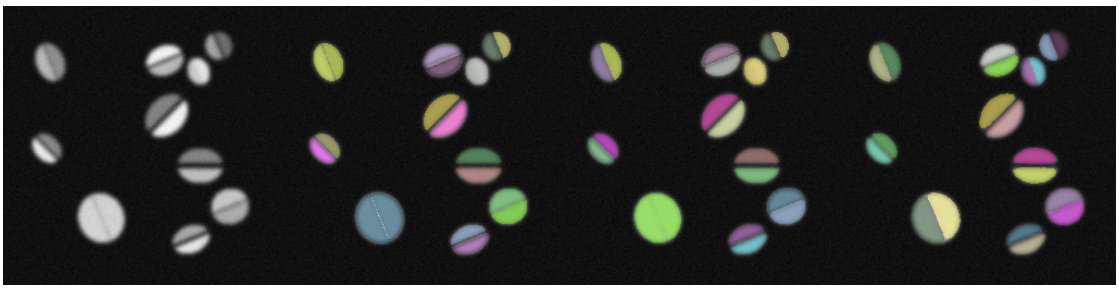


Figure 6.5: Segmentation results for the `TOY` dataset. Contains the original image, the segmentation of the 2-class U-Net, 3-class U-Net and StarDist model in this order.

For higher thresholds, StarDist gets worse quickly because the correct object shapes cannot be represented by the star-shaped polygons perfectly. Figure 6.5 confirms this assumption. The U-Net models wrongly merge objects while the StarDist model detects every object correctly but predicts non-perfect shapes.

Threshold	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90
U-Net (2-class)	.723	.664	.661	.661	.661	.657	.655	.652	.647
U-Net (3-class)	.878	.857	.857	.857	.857	.857	.857	.857	**.857**
StarDist	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**.965**	.411

Table 6.1: Average Precision (AP) of the evaluated models on the `TOY` dataset for different intersection over union (IoU) thresholds.

6.3.2 Data Science Bowl 2018 (STARDIST_DSB2018)

The `STARDIST_DSB2018` dataset contains real fluorescence microscopy images that are very diverse. The images have been annotated manually, but some annotations contain mistakes. The dataset consists of a total of 497 images (447 training, 50 testing) of varying sizes.

Again, three workflows were built that only differ from the TOY example workflows in the data loading. A screenshot of the workflow for the StarDist model is shown in Figure A.2.

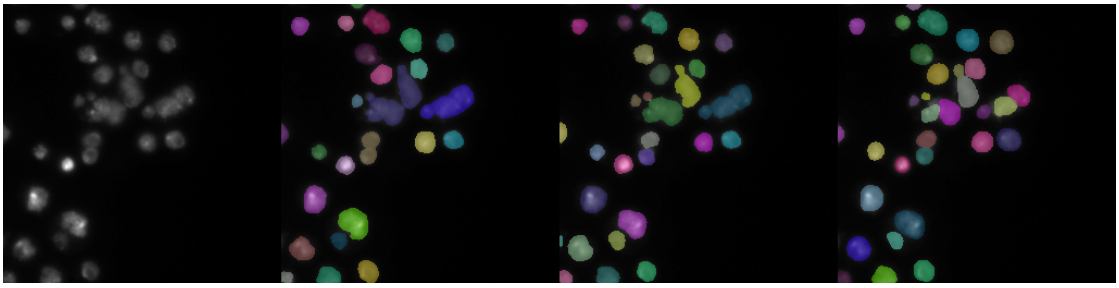


Figure 6.6: Segmentation results for the `STARDIST_DSB2018` dataset. Contains the original image, the segmentation of the 2-class U-Net, 3-class U-Net and StarDist model in this order.

Table 6.2 shows the Average Precision scores for each model. As for the `TOY` dataset, StarDist is superior for lower IoU thresholds but not for very high thresholds. Figure 6.6 shows the segmentation results of one selected image from the testing set. Again, StarDist is not able to reconstruct the exact shape performs well in recognizing all individual cells and separating them.

Threshold	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90
U-Net (2-class)	.722	.684	.655	.628	.600	.562	.495	.403	.299
U-Net (3-class)	.748	.704	.673	.643	.616	.575	**.512**	**.428**	**.311**
StarDist	**.820**	**.798**	**.766**	**.725**	**.672**	**.588**	.490	.350	.188

Table 6.2: Average Precision (AP) of the evaluated models on the `STARDIST_DSB2018` dataset for different intersection over union (IoU) thresholds.

6.3.3 Retraining StarDist

The developed framework can be used to reduce the training time and improve the accuracy of models by initializing the model weights with reasonable values and using transfer learning.

A new dataset was chosen to demonstrate this. The `PhC-C2DL-PSC` dataset from the *celltrackingchallenge*[1] consists of phase contrast microscopy images of stem cells and contains only four segmented images.

A KNIME workflow was build that reads a StarDist model, that was pretrained on `DSB2018`, and retrains it on the new dataset using the *Retrain StarDist Model* Component (see Figure A.4).

Figure 6.8 shows the training loss history of the StarDist model in KNIME. We can see that the loss of the pretrained model drops much quicker. The pretrained model has a much lower loss of about 0.08 after 178 batches than the randomly initialized model

[1] `http://celltrackingchallenge.net/2d-datasets/`

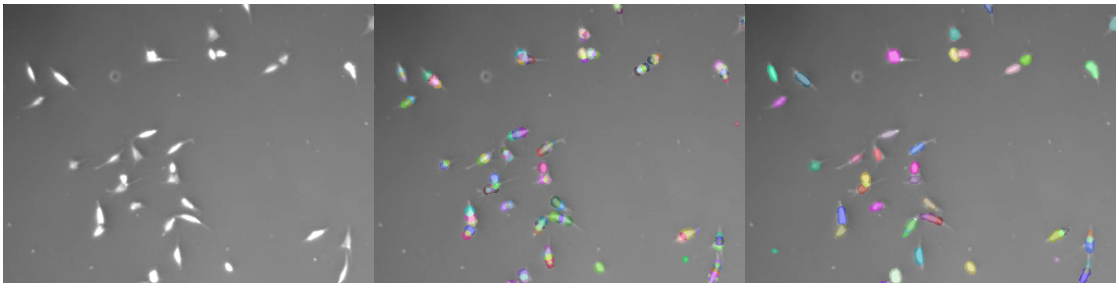
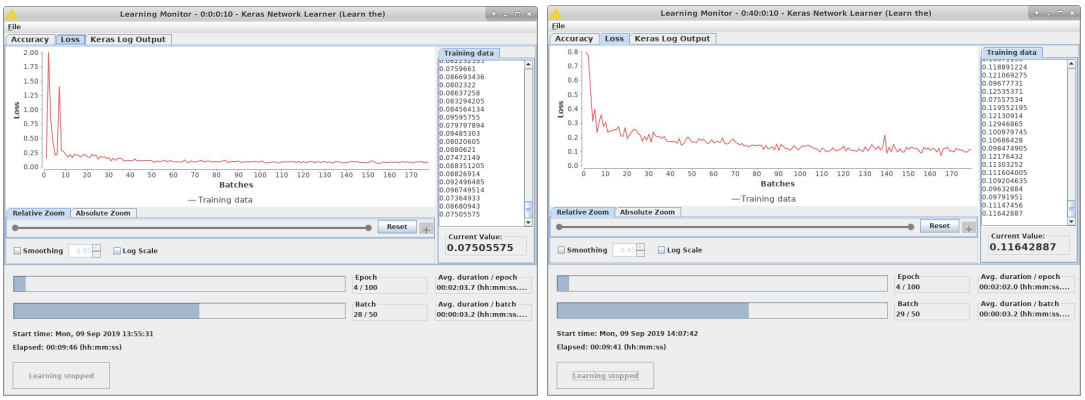


Figure 6.7: Segmentation results for the `PhC-C2DL-PSC` dataset. Contains the original image, the segmentation of the randomly initialized model and the pretrained model.



(a) Loss history with a pretrained model

(b) Loss history with a randomly initialized model

Figure 6.8: Loss history of a pretrained model and a randomly initialized model in KNIME.

which has a loss of about 0.12 after 179 batches. Both models trained for around ten minutes on an NVIDIA GeForce GTX 1080 Ti.

Figure 6.7 shows an example result for the trained models. We can see that the randomly initialized model (middle image) does not predict useful segmentations. The pretrained model, however, (right image) can detect and segment most of the cells. The pretrained model struggles to reproduce the cell shapes, but it does much better than the randomly initialized model, which predicts mostly circles.

6.3.4 StarDist on a Large Image

A workflow which executes the StarDist model on an image with a resolution of 3840×3840 was built. This workflow verifies that the framework enables execution of deep learning models on large images. The image was generated by repeating one of the testing examples of the `DSB2018` dataset, but images of this size are not unusual in some areas of microscopy like histopathology [Gur+09]. Note that the developed KNIME

Node *Tensor TensorFlow Executor* can also be applied to more-dimensional datasets where the memory issue appears much quicker.

The workflow (which is shown in Figure A.3) loads the image and applies the StarDist model trained in Section 6.3.2 on the image. The *Apply StarDist Model* Component executes the deep learning model by adding a context-map node to the computation graph, which executes the model on tiles of the image with a context. The *Apply StarDist Model (DL Executor)* Component executes the deep learning model on the whole image in one go. See also Section 6.2 for a description of the Nodes.

Experiments using the demonstrated workflows showed that executing the deep learning model on the large image in one go is not possible (on the tested hardware) while the tiled execution was possible (see Table 6.3). These results show that the developed framework enables an application of deep learning models for instance segmentation on large images that has not been possible before.

System	Laptop, 32 GB RAM, no GPU	Laptop, 32 GB RAM, 4 GB GPU	Workstation, 32 GB RAM, 11 GB GPU
Tiled *Apply StarDist Model*	✓ (RAM: $\sim$ 10 GB)	✓ (RAM: $\sim$ 8 GB)	✓ (RAM: $\sim$ 8 GB)
Non-Tiled *Apply StarDist Model (DL Executor)*	✗ (JVM crashed)	✗ (OOM Exception)	✗ (OOM Exception)

Table 6.3: Results of applying the StarDist model on a large image. The tiled execution did succeed on every configuration while the non-tiled execution failed on every configuration.

Chapter 7

Conclusion

This work investigated approaches to surpass the drawbacks of deep learning based segmentation for microscope images.

Using the StarDist model with the usual U-Net backbone and with the improved Res-U-Net backbone transfer learning on different microscopic datasets was evaluated. Five experiments were conducted to evaluate if transfer learning is applicable to microscopic image analysis. The experiments show that the choice of the pretraining dataset is essential and that transfer learning can improve the performance of the model on the target dataset. One fundamental observation is that transfer learning can reduce the required training time significantly. The reduced training time and the improved performance on datasets with only a few images make deep learning based segmentation methods more useful for microscopic image analysis.

To make deep learning based segmentation more accessible, a framework was developed that allows to train and apply deep learning models on images without programming knowledge and with simple usage patterns. The framework is easy to use (by reusing components) and extensible (by creating new components, KNIME Nodes, or lower-level operations). It was demonstrated that this framework can be used to train deep learning models model from scratch, retrain models on new data and apply models on arbitrarily sized images.

Additionally, an improvement to the StarDist model was proposed and evaluated. The backbone consisting of residual blocks was found to perform better on the more complex `DSB2018` dataset while not reducing the performance on another dataset. It has been shown that this performance increase is due to the residuals and not the increased model depth.

Future Work In future work methods to work with images that have a higher dimensionality and a varying number of channels need to be evaluated. Models that were pretrained on a dataset with one channel cannot be applied easily to images with multiple channels. Especially in fluorescence microscopy, the channels can contain essential information that needs to be handled appropriately by the model. It is not clear what pretraining is required for a fluorescence dataset with specific staining.

Additionally, other methods of transfer learning need to be evaluated on various microscope images. The related works chapter lists multiple approaches to improve transfer learning of deep networks. It remains to be seen if they can be applied successfully to microscope images.

Bibliography

[Aba+16] Martín Abadi et al. "TensorFlow: a system for large-scale machine learning". In: *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 265–283. ISBN: 978-1-931971-33-1. URL: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

[Arb+14] Pablo Arbelaez et al. "Multiscale combinatorial grouping". In: *2014 IEEE conference on computer vision and pattern recognition*. IEEE, June 2014. DOI: 10.1109/cvpr.2014.49. URL: https://doi.org/10.1109/cvpr.2014.49.

[axe] axelle b. *Cute kittens and adventurous*. License: CC0 Public Domain (https://creativecommons.org/publicdomain/zero/1.0/). URL: https://www.publicdomainpictures.net/en/view-image.php?image=150567&picture=cute-kittens-and-adventurous (visited on 09/04/2019).

[Ber+09] Michael R. Berthold et al. "KNIME - the konstanz information miner". In: *ACM SIGKDD explorations newsletter* 11.1 (Nov. 2009), p. 26. DOI: 10.1145/1656274.1656280. URL: https://doi.org/10.1145/1656274.1656280.

[BHL15] Michael Boutros, Florian Heigwer, and Christina Laufer. "Microscopy-based high-content screening". In: *Cell* 163.6 (Dec. 2015), pp. 1314–1325. DOI: 10.1016/j.cell.2015.11.007. URL: https://doi.org/10.1016/j.cell.2015.11.007.

[BM09] Suzanne Bell and Keith Morris. *An introduction to microscopy*. CRC Press, Oct. 2009. DOI: 10.1201/b15738. URL: https://doi.org/10.1201/b15738.

[Boo18] Booz Allen and Kaggle. *2018 Data Science Bowl - Data | Kaggle*. 2018. URL: https://www.kaggle.com/c/data-science-bowl-2018/data (visited on 08/01/2019).

[Cho+15] François Chollet et al. *Keras*. 2015. URL: https://keras.io.

[Cor+16] Marius Cordts et al. "The cityscapes dataset for semantic urban scene understanding". In: *2016 IEEE conference on computer vision and pattern recognition (CVPR)*. IEEE, June 2016. DOI: 10.1109/cvpr.2016.350. URL: https://doi.org/10.1109/cvpr.2016.350.

[Dar] Dartmouth College Electron Microscope Facility. *Pollen from a variety of common plants*. License: Public domain. URL: https://upload.wikimedia.org/wikipedia/commons/a/a4/Misc_pollen.jpg (visited on 09/04/2019).

[DB16] Christian Dietz and Michael R. Berthold. "KNIME for open-source bioimage analysis: a tutorial". In: *Focus on bio-image informatics*. Springer International Publishing, 2016, pp. 179–197. DOI: 10.1007/978-3-319-28549-8_7. URL: https://doi.org/10.1007/978-3-319-28549-8_7.

[Eco+16] Michael N Economo et al. "A platform for brain-wide imaging and reconstruction of individual neurons". In: *eLife* 5 (Jan. 2016). DOI: 10.7554/elife.10566. URL: https://doi.org/10.7554/elife.10566.

[Fal+18] Thorsten Falk et al. "U-net: deep learning for cell counting, detection, and morphometry". In: *Nature methods* 16.1 (Dec. 2018), pp. 67–70. DOI: 10.1038/s41592-018-0261-2. URL: https://doi.org/10.1038/s41592-018-0261-2.

[Gir+14] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *2014 IEEE conference on computer vision and pattern recognition*. IEEE, June 2014. DOI: 10.1109/cvpr.2014.81. URL: https://doi.org/10.1109/cvpr.2014.81.

[Gur+09] M.N. Gurcan et al. "Histopathological image analysis: a review". In: *IEEE reviews in biomedical engineering* 2 (2009), pp. 147–171. DOI: 10.1109/rbme.2009.2034865. URL: https://doi.org/10.1109/rbme.2009.2034865.

[Har+14] Bharath Hariharan et al. "Simultaneous detection and segmentation". In: *Computer vision – ECCV 2014*. Springer International Publishing, 2014, pp. 297–312. DOI: 10.1007/978-3-319-10584-0_20. URL: https://doi.org/10.1007/978-3-319-10584-0_20.

[He+16] Kaiming He et al. "Deep residual learning for image recognition". In: *2016 IEEE conference on computer vision and pattern recognition (CVPR)*. IEEE, June 2016. DOI: 10.1109/cvpr.2016.90. URL: https://doi.org/10.1109/cvpr.2016.90.

[He+17] Kaiming He et al. "Mask r-CNN". In: *2017 IEEE international conference on computer vision (ICCV)*. IEEE, Oct. 2017. DOI: 10.1109/iccv.2017.322. URL: https://doi.org/10.1109/iccv.2017.322.

[HGD18] Kaiming He, Ross B. Girshick, and Piotr Dollár. "Rethinking imagenet pre-training". In: *CoRR* abs/1811.08883 (Nov. 2018). arXiv: 1811.08883. URL: http://arxiv.org/abs/1811.08883.

[HHS17] Zeeshan Hayder, Xuming He, and Mathieu Salzmann. "Boundary-aware instance segmentation". In: *2017 IEEE conference on computer vision and pattern recognition (CVPR)*. IEEE, July 2017. DOI: 10.1109/cvpr.2017.70. URL: https://doi.org/10.1109/cvpr.2017.70.

[Hua+18] Xun Huang et al. “Multimodal unsupervised image-to-image translation”. In: *Computer vision – ECCV 2018*. Springer International Publishing, 2018, pp. 179–196. DOI: 10.1007/978-3-030-01219-9_11. URL: https://doi.org/10.1007/978-3-030-01219-9_11.

[Hui04] J. Huisken. “Optical sectioning deep inside live embryos by selective plane illumination microscopy”. In: *Science* 305.5686 (Aug. 2004), pp. 1007–1009. DOI: 10.1126/science.1100035. URL: https://doi.org/10.1126/science.1100035.

[KB15] Diederik P. Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *3rd international conference on learning representations, ICLR 2015, san diego*. May 2015. URL: http://arxiv.org/abs/1412.6980.

[Kel+11] P. J. Keller et al. “Digital scanned laser light-sheet fluorescence microscopy (DSLM) of zebrafish and drosophila embryonic development”. In: *Cold spring harbor protocols* 2011.10 (Oct. 2011), pdb.prot065839–pdb.prot065839. DOI: 10.1101/pdb.prot065839. URL: https://doi.org/10.1101/pdb.prot065839.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., Dec. 2012, pp. 1097–1105. URL: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.

[KSL19] Simon Kornblith, Jonathon Shlens, and Quoc V Le. “Do better imagenet models transfer better?” In: *2019 IEEE conference on computer vision and pattern recognition (CVPR)*. IEEE, June 2019.

[LBK17] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. “Unsupervised image-to-image translation networks”. In: *Advances in neural information processing systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., Dec. 2017, pp. 700–708. URL: https://papers.nips.cc/paper/6672-unsupervised-image-to-image-translation-networks.

[LeC+99] Yann LeCun et al. “Object recognition with gradient-based learning”. In: *Shape, contour and grouping in computer vision*. Springer Berlin Heidelberg, 1999, pp. 319–345. DOI: 10.1007/3-540-46805-6_19. URL: https://doi.org/10.1007/3-540-46805-6_19.

[Lin+14] Tsung-Yi Lin et al. “Microsoft COCO: common objects in context”. In: *Computer vision – ECCV 2014*. Springer International Publishing, 2014, pp. 740–755. DOI: 10.1007/978-3-319-10602-1_48. URL: https://doi.org/10.1007/978-3-319-10602-1_48.

[Lin+17a] Tsung-Yi Lin et al. “Feature pyramid networks for object detection”. In: *2017 IEEE conference on computer vision and pattern recognition (CVPR)*. IEEE, July 2017. DOI: 10.1109/cvpr.2017.106. URL: https://doi.org/10.1109/cvpr.2017.106.

[Lin+17b] Tsung-Yi Lin et al. “Focal loss for dense object detection”. In: *2017 IEEE international conference on computer vision (ICCV)*. IEEE, Oct. 2017. DOI: 10.1109/iccv.2017.324. URL: https://doi.org/10.1109/iccv.2017.324.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *2015 IEEE conference on computer vision and pattern recognition (CVPR)*. IEEE, June 2015. DOI: 10.1109/cvpr.2015.7298965. URL: https://doi.org/10.1109/cvpr.2015.7298965.

[Mei+09] Erik Meijering et al. “Tracking in cell and developmental biology”. In: *Seminars in cell & developmental biology* 20.8 (Oct. 2009), pp. 894–902. DOI: 10.1016/j.semcdb.2009.07.004. URL: https://doi.org/10.1016/j.semcdb.2009.07.004.

[Mei12] Erik Meijering. “Cell segmentation: 50 years down the road [life sciences]”. In: *IEEE signal processing magazine* 29.5 (Sept. 2012), pp. 140–145. DOI: 10.1109/msp.2012.2204190. URL: https://doi.org/10.1109/msp.2012.2204190.

[Ngi+18] Jiquan Ngiam et al. “Domain adaptive transfer learning with specialist models”. In: *Corr* abs/1811.07056 (Nov. 2018). arXiv: 1811.07056. URL: http://arxiv.org/abs/1811.07056.

[NH10] Vinod Nair and Geoffrey E. Hinton. “Rectified linear units improve restricted boltzmann machines”. In: *Proceedings of the 27th international conference on international conference on machine learning*. ICML’10. Omnipress, 2010, pp. 807–814. ISBN: 978-1-60558-907-7.

[Oqu+14] Maxime Oquab et al. “Learning and transferring mid-level image representations using convolutional neural networks”. In: *2014 IEEE conference on computer vision and pattern recognition*. IEEE, June 2014. DOI: 10.1109/cvpr.2014.222. URL: https://doi.org/10.1109/cvpr.2014.222.

[OS18] Hugo Oliveira and Jefersson dos Santos. “Deep transfer learning for segmentation of anatomical structures in chest radiographs”. In: *2018 31st SIBGRAPI conference on graphics, patterns and images (SIBGRAPI)*. IEEE, Oct. 2018. DOI: 10.1109/sibgrapi.2018.00033. URL: https://doi.org/10.1109/sibgrapi.2018.00033.

[Ots79] Nobuyuki Otsu. “A threshold selection method from gray-level histograms”. In: *IEEE transactions on systems, man, and cybernetics* 9.1 (Jan. 1979), pp. 62–66. DOI: 10.1109/tsmc.1979.4310076. URL: https://doi.org/10.1109/tsmc.1979.4310076.

[PY10] Sinno Jialin Pan and Qiang Yang. "A survey on transfer learning". In: *IEEE transactions on knowledge and data engineering* 22.10 (Oct. 2010), pp. 1345–1359. DOI: 10.1109/tkde.2009.191. URL: https://doi.org/10.1109/tkde.2009.191.

[Red+16] Joseph Redmon et al. "You only look once: unified, real-time object detection". In: *2016 IEEE conference on computer vision and pattern recognition (CVPR)*. IEEE, June 2016. DOI: 10.1109/cvpr.2016.91. URL: https://doi.org/10.1109/cvpr.2016.91.

[Ren+15] Shaoqing Ren et al. "Faster r-cnn: towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 91–99. URL: https://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: convolutional networks for biomedical image segmentation". In: *Lecture notes in computer science*. Springer International Publishing, 2015, pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28. URL: https://doi.org/10.1007/978-3-319-24574-4_28.

[RSF19] Artem Rozantsev, Mathieu Salzmann, and Pascal Fua. "Beyond sharing weights for deep domain adaptation". In: *IEEE transactions on pattern analysis and machine intelligence* 41.4 (Apr. 2019), pp. 801–814. DOI: 10.1109/tpami.2018.2814042. URL: https://doi.org/10.1109/tpami.2018.2814042.

[Rue+17] Curtis T. Rueden et al. "ImageJ2: ImageJ for the next generation of scientific image data". In: *BMC bioinformatics* 18.1 (Nov. 2017). DOI: 10.1186/s12859-017-1934-z. URL: https://doi.org/10.1186/s12859-017-1934-z.

[Rus+15] Olga Russakovsky et al. "ImageNet large scale visual recognition challenge". In: *International journal of computer vision* 115.3 (Apr. 2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y. URL: https://doi.org/10.1007/s11263-015-0816-y.

[Sch+12] Johannes Schindelin et al. "Fiji: an open-source platform for biological-image analysis". In: *Nature methods* 9.7 (June 2012), pp. 676–682. DOI: 10.1038/nmeth.2019. URL: https://doi.org/10.1038/nmeth.2019.

[Sch+18] Uwe Schmidt et al. "Cell detection with star-convex polygons". In: *Medical image computing and computer assisted intervention – MICCAI 2018*. Springer International Publishing, 2018, pp. 265–273. DOI: 10.1007/978-3-030-00934-2_30. URL: https://doi.org/10.1007/978-3-030-00934-2_30.

[Ser+14] Pierre Sermanet et al. "Overfeat: integrated recognition, localization and detection using convolutional networks". In: *3rd international conference on learning representations, ICLR 2014, banff*. Apr. 2014. URL: http://arxiv.org/abs/1312.6229.

[SKS09] David Svoboda, Michal Kozubek, and Stanislav Stejskal. "Generation of digital phantoms of cell nuclei and simulation of image formation in 3d image cytometry". In: *Cytometry part a* 75A.6 (June 2009), pp. 494–509. DOI: 10.1002/cyto.a.20714. URL: https://doi.org/10.1002/cyto.a.20714.

[SRE12] Caroline A Schneider, Wayne S Rasband, and Kevin W Eliceiri. "NIH image to ImageJ: 25 years of image analysis". In: *Nature methods* 9.7 (June 2012), pp. 671–675. DOI: 10.1038/nmeth.2089. URL: https://doi.org/10.1038/nmeth.2089.

[SS13] R. Summerbell and J. Scott. *Microsporum audouinii macroconidium in bright field microscopy*. License: CC0 Public Domain (https://creativecommons.org/publicdomain/zero/1.0/). 2013. URL: https://commons.wikimedia.org/wiki/File:Microsporum_audouinii_macroconidium.jpg (visited on 09/04/2019).

[Ste+14] Johannes Stegmaier et al. "Fast segmentation of stained nuclei in terabyte-scale, time resolved 3d microscopy image stacks". In: *PLoS ONE* 9.2 (Feb. 2014). Ed. by Konradin Metze, e90036. DOI: 10.1371/journal.pone.0090036. URL: https://doi.org/10.1371/journal.pone.0090036.

[Ste+16] Johannes Stegmaier et al. "Real-time three-dimensional cell segmentation in large-scale microscopy data of developing embryos". In: *Developmental cell* 36.2 (Jan. 2016), pp. 225–240. DOI: 10.1016/j.devcel.2015.12.028. URL: https://doi.org/10.1016/j.devcel.2015.12.028.

[SZ15] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *3rd international conference on learning representations, ICLR 2015, san diego*. May 2015. URL: http://arxiv.org/abs/1409.1556.

[Sze+15] Christian Szegedy et al. "Going deeper with convolutions". In: *2015 IEEE conference on computer vision and pattern recognition (CVPR)*. IEEE, June 2015. DOI: 10.1109/cvpr.2015.7298594. URL: https://doi.org/10.1109/cvpr.2015.7298594.

[Ulm+17] Vladimír Ulman et al. "An objective comparison of cell-tracking algorithms". In: *Nature methods* 14.12 (Oct. 2017), pp. 1141–1152. DOI: 10.1038/nmeth.4473. URL: https://doi.org/10.1038/nmeth.4473.

[Wei+18] Martin Weigert et al. "Content-aware image restoration: pushing the limits of fluorescence microscopy". In: *Nature methods* 15.12 (Nov. 2018), pp. 1090–1097. DOI: 10.1038/s41592-018-0216-7. URL: https://doi.org/10.1038/s41592-018-0216-7.

[Wik19a] Wikipedia contributors. *Confirmation bias — Wikipedia, the free encyclopedia*. 2019. URL: https://en.wikipedia.org/w/index.php?title=Confirmation_bias&oldid=888916268 (visited on 04/02/2019).

[Wik19b] Wikipedia contributors. *Fluorescence microscope — Wikipedia, the free encyclopedia*. 2019. URL: https://en.wikipedia.org/w/index.php?title=Fluorescence_microscope&oldid=909817261 (visited on 09/02/2019).

[Wik19c] Wikipedia contributors. *Microscope — Wikipedia, the free encyclopedia*. 2019. URL: https://en.wikipedia.org/w/index.php?title=Microscope&oldid=903671783 (visited on 07/22/2019).

[Yos+14] Jason Yosinski et al. "How transferable are features in deep neural networks?" In: *Advances in neural information processing systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., Dec. 2014, pp. 3320–3328. URL: https://papers.nips.cc/paper/5347-how-transferable-are-features-in-deep-neural-networks.

[ZEI15] ZEISS Microscopy from Germany. *Mitotic LLC-PK1 cells, fluorescence microscopy*. License: CC BY 2.0 (https://creativecommons.org/licenses/by/2.0/), no changes were made. 2015. URL: https://commons.wikimedia.org/wiki/File:Mitotic_LLC-PK1_cells,_fluorescence_microscopy_(23700644352).jpg (visited on 09/04/2019).

[Zha+19] Jing Zhang et al. "Recent advances in transfer learning for cross-dataset visual recognition". In: *ACM computing surveys* 52.1 (Feb. 2019), pp. 1–38. DOI: 10.1145/3291124. URL: https://doi.org/10.1145/3291124.

[Zhu+17] Jun-Yan Zhu et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks". In: *2017 IEEE international conference on computer vision (ICCV)*. IEEE, Oct. 2017. DOI: 10.1109/iccv.2017.244. URL: https://doi.org/10.1109/iccv.2017.244.

Appendix A

KNIME Workflows

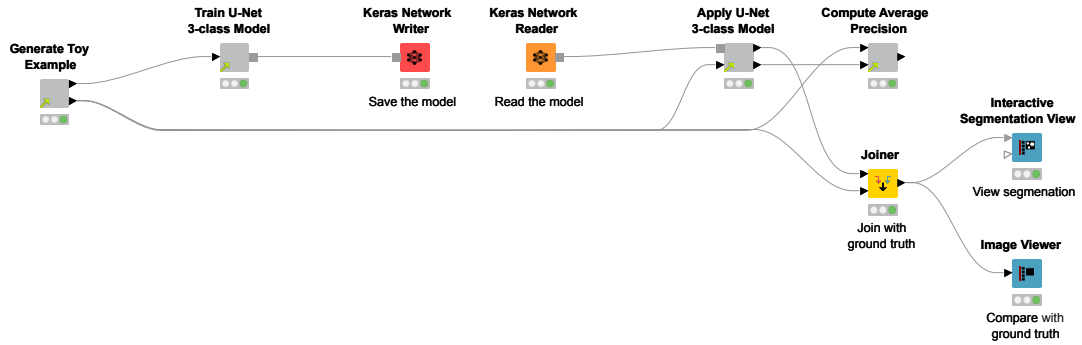


Figure A.1: KNIME Workflow for training and evaluating a U-Net 3-class model on the `TOY` dataset.

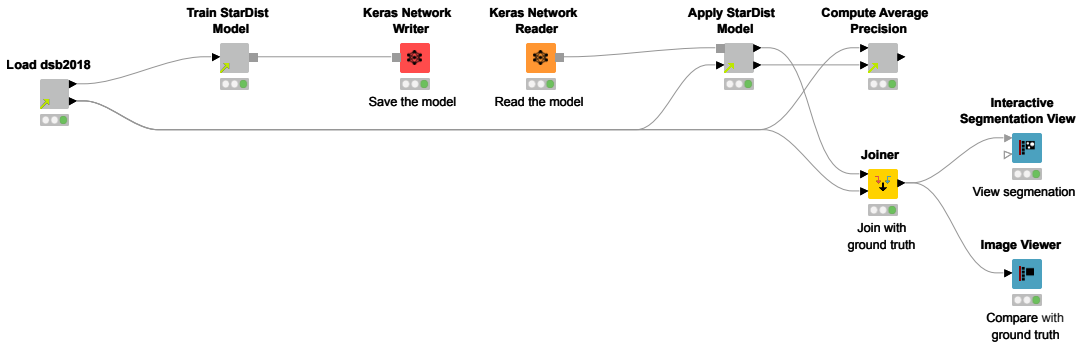


Figure A.2: KNIME Workflow for training and evaluating a StarDist model on the `STARDIST_DSB2018` dataset.

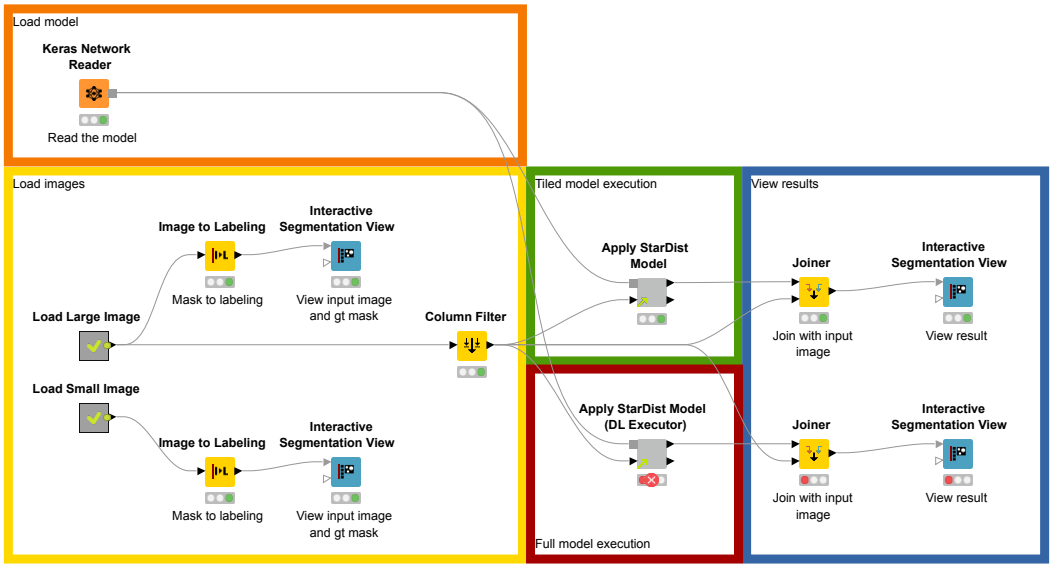


Figure A.3: KNIME Workflow for applying a StarDist model on a large image.

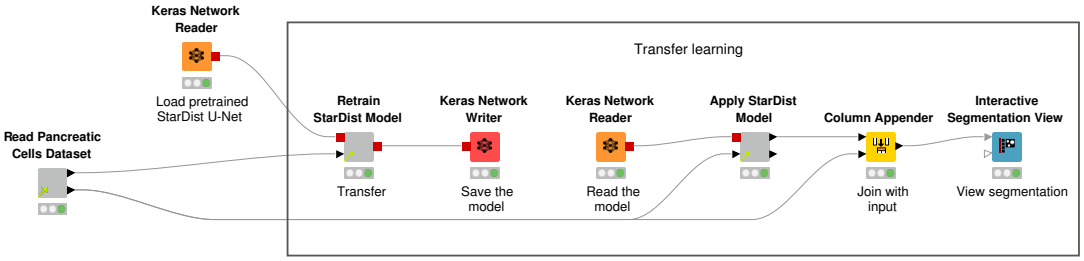
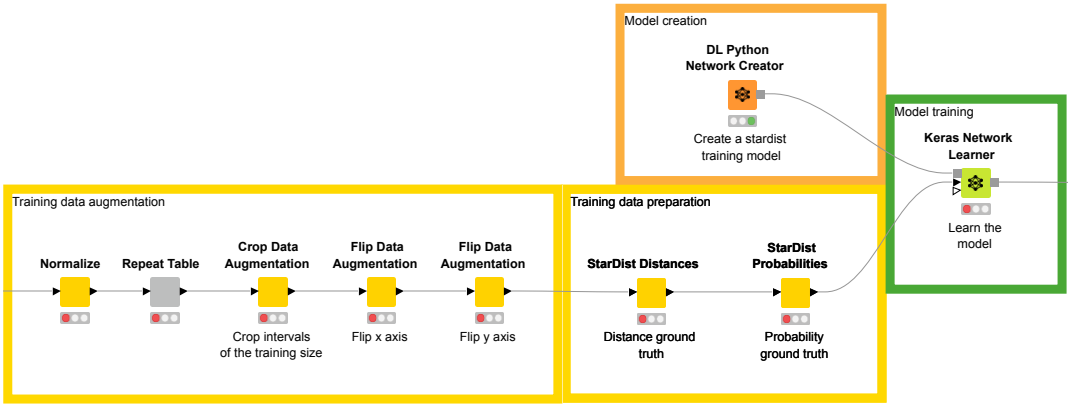
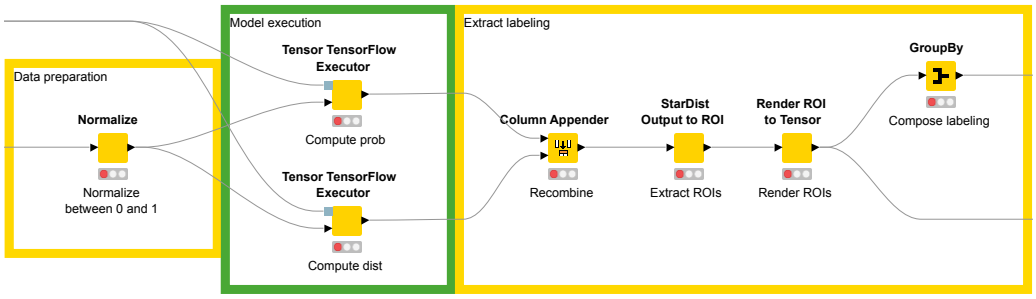


Figure A.4: KNIME Workflow for retraining a StarDist model on the `PhC-C2DL-PSC` dataset.



(a) Selected parts of the *Train StarDist Model* Component.



(b) Selected parts of the *Apply StarDist Model* Component.

Figure A.5: Parts of the training and inference workflows for the StarDist model. The normalization, data augmentation, training data preparation, model execution, and labeling extraction is done by adding operations to the Tensor Processing computation graph.

Appendix B

Additional Results

This appendix contains plots for a few additional experiments that were executed. The results are not described or interpreted.

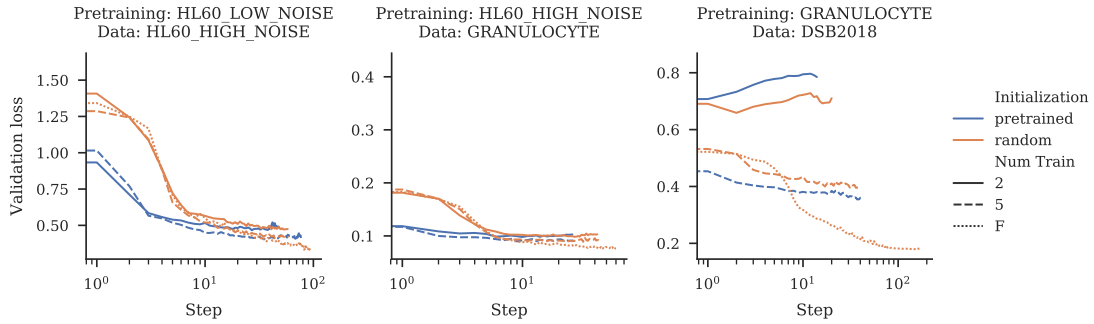


Figure B.1: Validation loss history of models that have been pretrained on a simulated dataset compared with the history of models that have been initialized randomly. The pretrained models achieve a lower loss quicker.

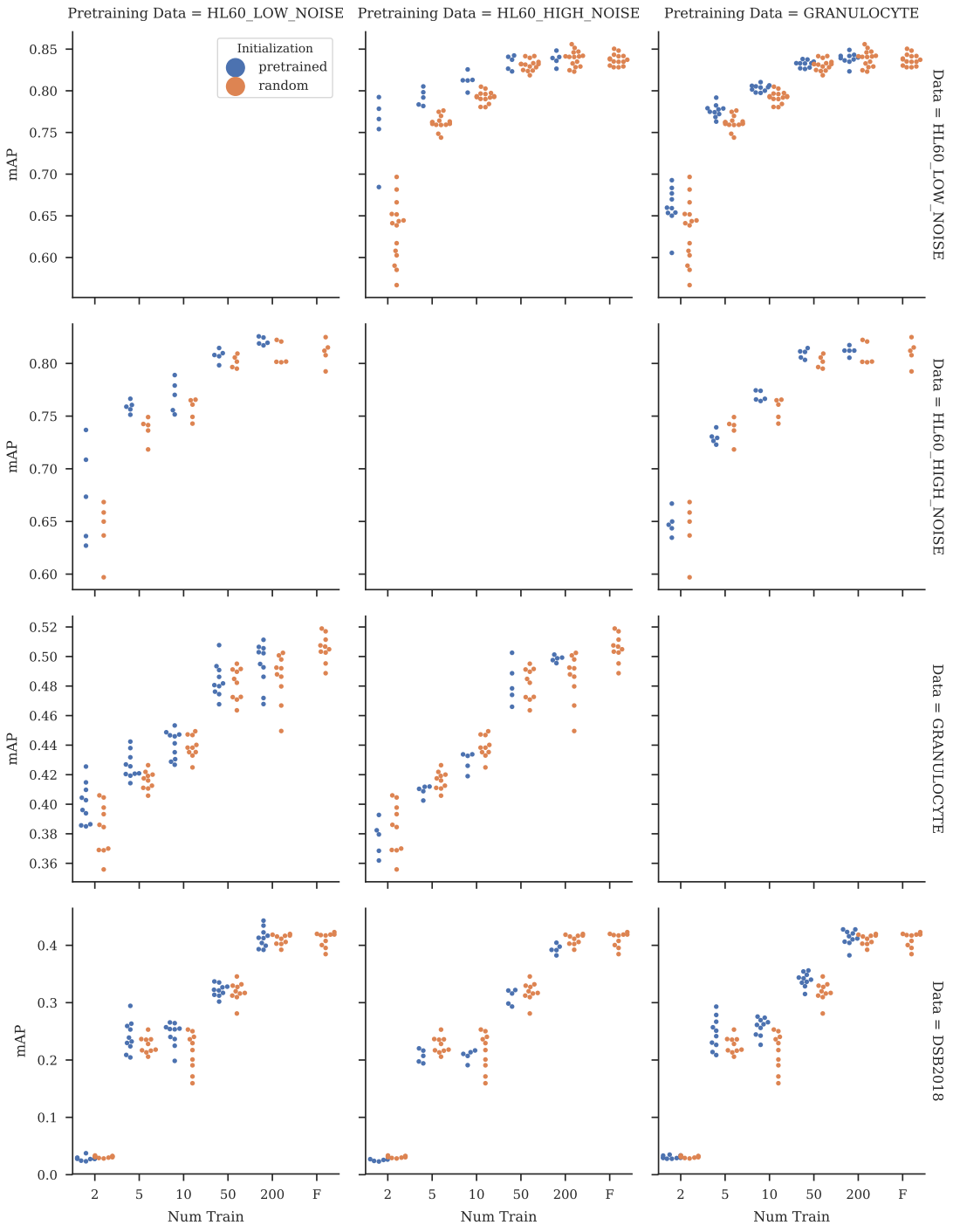


Figure B.2: Mean AP for models that have been pretrained on a simulated dataset compared with models that have been trained with random initialization. All models use a StarDist head and a simple U-Net backbone.

Appendix C

Installation Instructions

This appendix describes how to setup KNIME Analytics Platform to use the Instance Segmentation Framework KNIME Nodes from Chapter 6.

1. Download KNIME Analytics Platform: `https://www.knime.com/downloads`.
2. Install and setup the KNIME Python Integration as describe here: `https://docs.knime.com/latest/python_installation_guide/index.html`.
3. Install and setup the KNIME Keras Integration as described here: `https://docs.knime.com/latest/deep_learning_installation_guide/index.html#keras-integration`.
4. Install and setup the KNIME TensorFlow Integration as described here: `https://docs.knime.com/2018-12/deep_learning_installation_guide/index.html#tensorflow-integration`.
5. Add the *Instance Segmentation* update site:
   Go to `File` → `Preferences` → `Install/Update` → `Available Software Sites` and add the URL `https://dev.b-wilhelm.de/instance-seg/`. Also deactivate the *Community Extensions* update site because it contains conflicting plugins.
6. Install the special dependencies by installing every plugin from the added update site: Go to `Help` → `Install New Software...`, select the added update site, deselect `Group items by category` and install every listed extension.
7. Import the `Instance Segmentation Framework`: Download the framework from `https://drive.google.com/open?id=1qNFsEof4Iyu8k4SaJ-RIVOnccdI5c14-` and import it into KNIME Analytics Platform with `File` → `Import KNIME Workflow...`.

The code can be found at
`https://drive.google.com/open?id=1OjQld8cKPG1osjyrmWJzZwK9_mZs75mr`.