Zero-Cost Deep Learning to Enhance Microscopy

Johanna Maria Jukkala

1800207 jjukkala@abo.fi

Master's Thesis

Åbo Akademi University

Department of Biosciences

Faculty of Science and Engineering

19.05.2022

Master´s degree in Biomedical Imaging

Specialization Theme: Light Microscopy Imaging

Supervisors:

1: Guillaume Jacquemet, Ph.D.

Responsible professor:

Diana Toivola, Associate Professor

ÅBO AKADEMI UNIVERSITY

Department of Biosciences

Faculty of Science and Engineering

JOHANNA JUKKALA

ZERO-COST DEEP-LEARNING APPROACHES TO ENHANCE MICROSCOPY

Master´s thesis Plan, 65 pp. Appendix 6 pp.

Biomedical Imaging

May 2022

Abstract: Combining microscopy image acquisition and deep learning improves image processing and analytics. However, deep learning requires knowledge of information technology and expensive hardware. Also, proper training of the network is essential for the successful prediction of unseen images, and understanding the limits of network training is important. The aim of this Master's thesis is to make free deep learning tools accessible for users to use, learn and share these methods in the field of microscopy image analysis. We created user-friendly Google Colaboratory notebooks for microscopy image segmentation (StarDist), restoration (CARE), and denoising (N2V). These notebooks are an easy and free introduction to deep learning but the limited Graphical Processing Unit (GPU) provided inhibits large-scale use. This Master's thesis is a part of a collaboration project called ZeroCostDL4Mic.

KEYWORDS: Deep Learning, Convolutional Neural Networks, Content-aware image Restoration, Noise2VOID, StarDist

LIST OF ABBREVIATIONS

AI	=	Artificial Intelligence
CARE	=	Content-aware Image restoration
CNN	=	Convolution neural networks
CPU	=	Central Processing Unit
DL	=	Deep Learning
GPU	=	Graphical Processing Unit
MIP	=	Maximum Intensity Projection
ML	=	Machine Learning
N2V	=	Noise2VOID
NN	=	Neural networks
NRMSE	=	Normalized Root-Mean-Square Error
PSNR	=	Peak signal-to-noise ratio
QC	=	Quality Control
SIM	=	Structured illumination microscopy
SSIM	=	Structural Similarity Index Measure
SNR	=	Signal-to-noise ratio
TPU	=	Tensor Processing Unit

Table of Contents

1. Introduction

1.1 Deep learning

Artificial intelligence (AI) is common nowadays (Figure 1). It can be used to predict stock markets or translate speech on Youtube in real-time. The common thing for all tasks is that they contain a lot of data and it is analyzed by mimicking human intelligence.

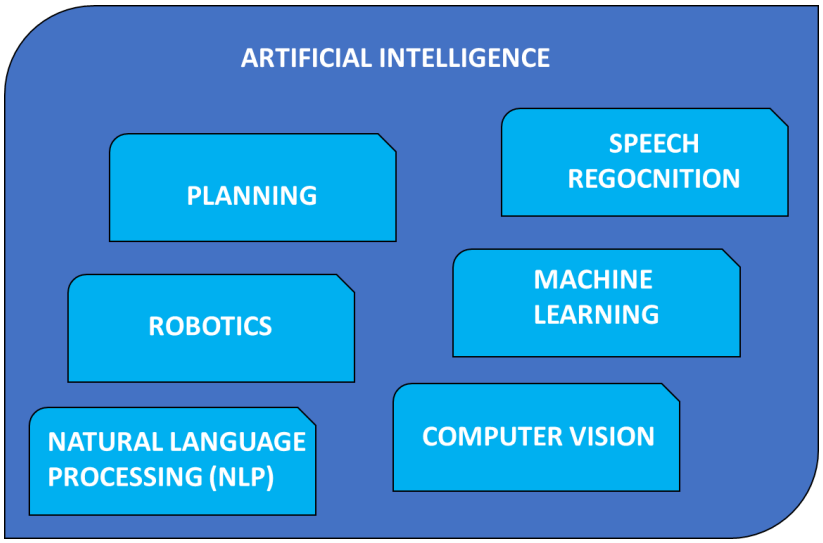


Figure 1. Artificial intelligence is utilized in many areas.

Artificial intelligence contains sub-categories: Machine learning (ML) and deep learning (DL) (Figure 2). Most applications of machine learning contained four steps: data cleaning and preprocessing, feature extraction, model fitting, and evaluation (Angermueller et al., 2016). Preprocessing and data cleaning are time-consuming but the real bottleneck of ML is feature extraction.

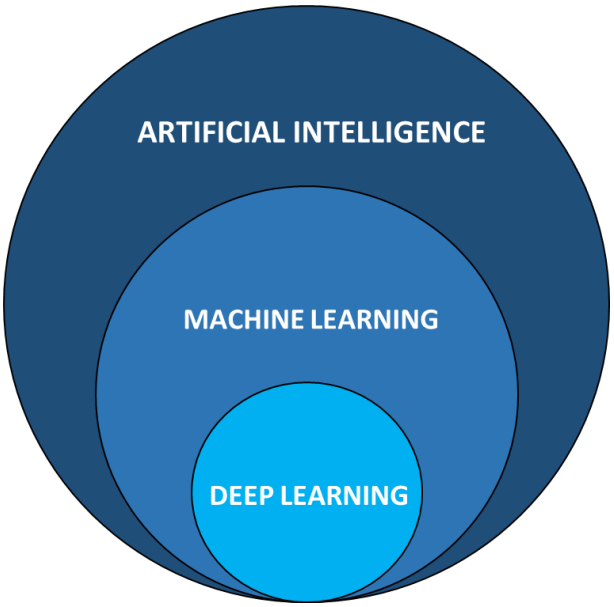


Figure 2: Even though though deep learning (DL) is more complex than machine learning (ML) they both are effective tools for analyzing various data which would otherwise consume time for humans. Together they are subsets of artificial intelligence (AI), which solves given problems by mimicking human logic.

The fundamental difference between humans and computers is the ability to recognize shapes and differences. Humans can easily tell if there is a cat or dog in the painting but the computer has serious difficulties. Therefore, features that contain the information about shapes must be taught to the computer. Unfortunately, an efficient ML model requires countless features and for high-dimensional images, feature extraction is more laborious (Angermueller et al., 2016).

After the invention of convolutional neural networks (CNNs), a new category of AI was formed (Figure 3). The key ability of deep learning (DL) is the ability to extract abstract features without teaching (Angermueller et al., 2016). For example, in ML humans teach that the concept of a car includes a motor, tires, and steering wheel. For DL the different images of cars are just shown and DL learns the concept of the car. Deep learning thus mimics human visual recognition.

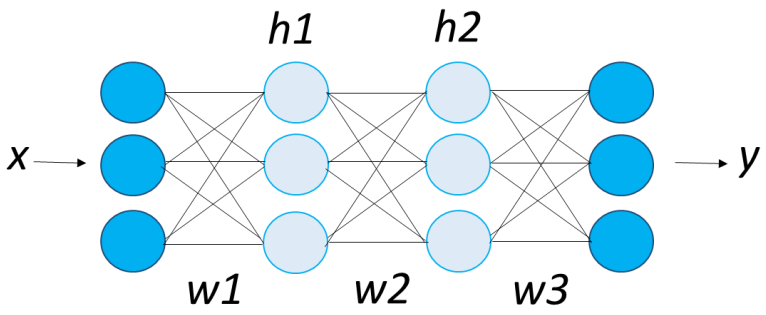


Figure 3: The general architecture of the neural network. The low SNR image *x* is restored to the high-quality *y* image. The network contain layers h_x and calculation weights w_x. The architecture may differ depending on the deep learning method. In this thesis all networks are UNET based architecture.

1.2 Deep learning and fluorescence microscopy

Fluorescence microscopy has developed and has become a common tool in modern laboratories. This means a massive number of images and datasets and a common trend is to have increased the number of public datasets. All this benefits deep learning and CNN has successfully been used for analyses of microscopy images. Denoising microscopy images have been done by CARE (Weigert et al., 2018), Noise2Void (Krull et al., 2019), and Noise2Self (Batson et al,. 2019). Predicted labeling of immunofluorescence (IF) images and electron micrograph (EM) images by fnet (one type of NN) (Ounkomol et al., 2018). The resolution of images was improved by SISR (Yang et al. 2018) and iSIM (Fang et al., 2019). Segmentation on cell nuclei has been done by StarDist (Schmidt et al., 2018) and U-net (Ronneberger et al., 2015). The classification has been done in U2OS cell line images by DeadNet (Richmond, 2017) and yeast cell images by DeepLoc (Kraus et al., 2017).

1.2.1 Image restoration

As fluorescence microscopies have become a general tool, several obstacles have been resolved. Despite the technological improvement of fluorescence microscopies, there are still fundamental issues which remain: signal-to-noise ratio (SNR) and resolution.

Imaging for an extended period causes photobleaching and the location of fluorophores is challenging to detect. Also, cells suffer during extended imaging sessions due to the high laser intensity used which will affect their behavior and may trigger their death (phototoxicity). One way to limit both issues is to decrease the laser intensity during the imaging, but this often leads to the acquisition of sub-optimal images (low signal-to-noise ratio). In fluorescence microscopy, image noise can be reduced by setting exposure time and intensity by careful sample preparation. Unfortunately, it might lead to sample degradation by phototoxicity and thus causing more problems (Belthangady and Royer, 2019).

This means that obtained microscopy images have always lost some irrecoverable information because of the physical restrictions. Compared to the ideal image they are degraded, incompleted, and convoluted. The common example of this is image noise, pixel-value quantization, and low-pass filtering. (Belthangady and Royer, 2019)

Reconstruction of the obtained microscopy images is one approach to overcome this loss of information. Traditionally microscopy images have been processed manually in image processing programs like ImageJ or BioimageXD. The common processes are noise removal by using algorithms like *blur*, *median* or *Gaussian Blur*, object detection, segmentation, and adjusting brightness, contrast, and colors. The user chooses the algorithm, input parameters and the software do the rest.

The problem with the traditional algorithms is the complexity of real-life images. These algorithms are designed based on the

previous information of image acquisition and statistical information on how for example image noise forms. In a certain context, they are efficient and fast but they struggle because they are not able to capture the whole statistical complexity of the microscopy images (Belthangady and Royer, 2019). Commonly the researcher faces this problem while processing microscopy images. For example, the traditional algorithm like *median* filter processes previously known information (Poisson statistics). It removes the image noise efficiently from the image but it also blurs the details which are important for the researcher.

Traditional algorithms process unseen images according to previous knowledge given by the user. (for example, the blur algorithm blurs the image given value). Deep learning studies the training images given by the user and processes the unseen mages based on that study. This ability makes deep learning more flexible than traditional algorithms. CNN processes datasets and thus utilizes the whole statistical complexity of images. (Belthangady and Royer, 2019). Also, deep learning can drastically improve the quality of low-signal-to-noise images and enable extended live microscopy using low laser power. Combining CNN with microscopy, biological samples can be treated and imaged more gently and still achieve low-noise images.

Content-aware image restoration (CARE) is a supervised deep learning method that uses convolutional neural networks to restore low SNR images. The network contains several layers called convolutional and pooling layers. First the network extracts the image features by convolutional layers followed by a pooling layer which simplifies all collected features and forms a simplified input image (like barcode). This simplified input is compared to the corresponding high-quality image. (Chamier et al., 2019) The trained network contains the information from all these simplified inputs and uses this information to restore new (but similar) unseen images. The training data contains pair images for training networks: high signal-to-noise ratio (SNR) images as

training images and low SNR images as target images. (Weigert et al. 2018)

Noise2VOID (N2V) is a denoising method for image restoration and it was created by A. Krull in 2018. The method utilizes blind spot networks but does not require pair images for the training phase. This is a self-supervised deep learning method where the user can use noisy images as input and target images. N2V uses a blind-spot network where the receptive field excludes the central pixel value (the convolutional receptive field includes all pixels). Excluding the central pixel value inhibits the network to learn pixel identity but still removes pixel-wise independent noise. (A. Krull et al., 2019)

1.2.2 Segmentation

Image segmentation splits the image in the background and foreground. Foreground contains all features (like cells or cell nuclei) which are above the threshold and the background is the rest of the image. There are two segmentation tasks: semantic segmentation and instance segmentation (Moen et al., 2019). Semantic segmentation labels each pixel semantically, meaning parts like this pixel belong to a cell, that pixel belongs to cytoplasm etc. However, Semantic segmentation may fail to separate the overlapping cells. Instance segmentation identifies the group of pixels instance of a class in the image (like do these pixels form a cell or not). (Moen et al., 2019) Top-down and bottom-up approaches are two different strategies to detect nuclei. Bottom-up strategy labels the pixels semantic classes (cell nuclei or to the background) and after that decides the instance of class. Top-down approach begins to segment the crude shape of each cell nuclei and after that refines the boundaries according to the possibility of shapes. (Schmidt et al., 2018)

1.2.3 Object detection and image classification

Well-known example of classification is the task whether the object in the image is cat or dog. In the biomedical field, classification is useful to identify different cell organs or cells in different cell cycles. Classification has great potential in healthcare and drug testing where the malignant cells or cancer cells are detected from tissues and may even detect cancer cells without need of chemical staining (Chen et al., 2016). Unsupervised method was used for cellular morphological phenotyping (Yao et al., 2016).

Searching the cells from the sample is the most basic task in microscopy. For the human eye, it is remotely easy to notice cells with different shapes and sizes. Unfortunately screening the multiple cell images manually consumes time and it is frustrating in the long run. There are applications for detecting cell-like segmentation by threshold (ImageJ) but the success of the detection is depending on the quality of the image. A challenging background of image, bleaching, uneven illumination, overlaying cells and non-common cell shapes increase the challenge of this task.

Object detection is a useful method for the detection of cells and cell nuclei from microscopy images. StarDist is a supervised segmentation method for cell nuclei detection created by Schmidt et al. in 2018. The method uses convolutional neural networks with U-net architecture created by Olaf Ronneberger, Philipp Fischer, and Thomas Brox in 2015 (Figure 4). The U-net architecture contains the ordinary convolutional layers but the pooling layers are replaced by the upsampling operators. This results in the larger resolution which is needed to detect cell nuclei boundaries. (Ronneberger et al., 2015). StarDist creates object candidates in two phases. First StarDist measures the distance *r* from the pixel to the assumed boundary and forms a star-convex polygon (the shape of possible nuclei) for each pixel. After that StarDist calculates the probability of the polygon. All object candidates are evaluated by non-maximum suppression

(NMS) to avoid duplicates and the final set of polygons are generated. (Schmidt et al., 2018)

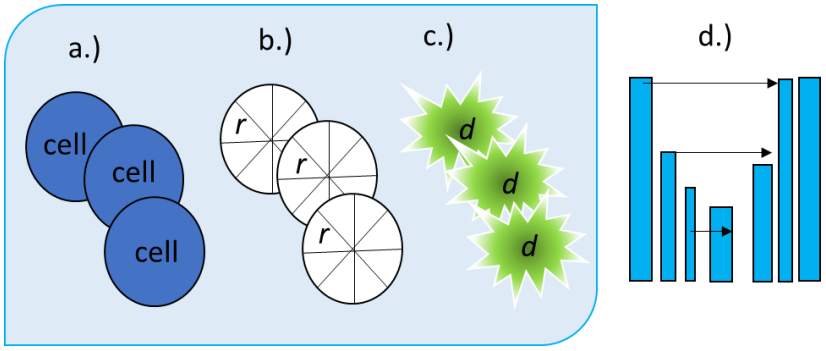


Figure. 4. General overflow of StarDist. The cell nucleus are segmented from the input image (a) by measuring the distance *r* of each cell nucleus (b) and probability of the nucleus *d* (c). The U-net architecture results in better resolution to detect boundaries (d). The probability of each object (*r* and *d* combined) is measured by NMS. The figure is inspired by the original paper (Schmidt et al., 2018).

1.3 Steps of Deep Learning

To use deep learning efficiently the four steps are needed to be considered: groundwork, training data, training the network and prediction. The groundwork is more like pre-step and others are actual process.

The first step is groundwork which includes the skills (programming, mathematics, software engineering) and hardware (invests, assembling, maintenance and upgrades). For researchers who have no previous knowledge or extra funding this step is the most difficult part. Moen raises this challenge in his article (Moen et al., 2019) and describes it as cultural barriers that inhibit the spreading of deep learning in biology labs. Training data is the foundation of a deep learning process. There is no training method which could overcome the failures of training data. Generating the training data is often time-

consuming (if supervised method is used) as it requires paired images (training images and ground truth image) and ground truth images often must be labeled manually or they are achieved by expensive microscopy. Premade training data are available but they might not be suitable for the experiment and may contain labeling errors. Training the network is a crucial part of how the trained network can be used. The challenge is to avoid overfitted models where the network learns too well and thus fails to predict the unseen data. Thus, the evaluation has a strong part of the training process. Another major challenge is the memory capacity as training requires a lot of computing power. Prediction is often quickly done and stuff. Training data, training the network and prediction are discussed in detail in section 1.4.

1.3.1 Groundwork

Both mathematics and biology are natural sciences but they operate sometimes quite far from each other. Deep learning is a collection of mathematical methods and algorithms but it is still considered as “a black box” C. (Belthangady and Royer., 2019). Deep learning is based on gradient descent but how deep learning decides certain outputs is still a mystery at a deep level even for mathematicians. Therefore, deep learning and algorithms may show opaque to biologists (Moen et al., 2019). As for good science, the researcher must provide methods and tools to prove his results (Nature Research. *Editorial policies*. 2020). If the researcher does not understand how deep learning works it is tempting to use the old methods.

Above the theoretical challenge is the technical issues: the need for specific hardware and power consumption of the hardware. Deep learning requires millions of parameters (Moen et al., 2019) and a high level of accuracy (Carneiro et al., 2018) to execute properly. Not only it stresses a regular computer a simple execution would take a long time. For example, in the medical field time is an important factor when the patient is diagnosed; delays may cost lives (Carneiro et al., 2018). For researchers,

time is also a crucial resource. Then regular computers contain a central processing unit (CPU) which fits poorly to the execution of deep learning. Graphics processing units (GPU) or Tensor processing units (TPU) perform much faster than regular CPUs but they are costly and often not a default option in regular computers. The solution is to either buy and assemble it yourself or perform cloud computing where the user uploads the training data to the cloud and GPU is provided by a company.

Assembling deep learning needs specific software and programming skills. Building the CNNs requires programming and Python is a common programming language but also R and C++ are used (Angermueller et al., 2016). Fortunately, the need for customization of deep learning for biological science has been recognized in computer vision communities (Belthangady and Royer, 2019). The pipeline of deep learning has been modified and the development aims now to more interactive graphical tools which help to explain how deep learning works, how the researcher can control the process, and how results are achieved (Belthangady and Royer, 2019). These tools are discussed in section 1.5.

1.4 Training data, training the network and prediction

1.4.1 Training data

From all components of deep learning the training, data is the most critical part of all processes (Belthangady and Royer, 2019) (Moen et al., 2019). Inappropriate training data affects negatively to the training process and trained networks produce hallucinations and other mistakes (discussed later in the next section). The proper training data for supervised deep learning methods should be well-annotated, error-free, large enough, normalized, publicly available, and specific for certain biological problems (Moen et al., 2019).

In supervised deep learning training, the training data contains usually paired images: training image and ground truth image (Figure 5). The network compares these paired images together and finds the parameters by which the network results in the output image as similar as possible to the ground truth image.

The generating the training data is time consuming and laborious especially when supervised deep learning methods are used. Sometimes achieving images might be challenging. Microscopies are not available, or the biological samples are scarce and there is not enough for the large-scale imaging. Researchers may seek images from his colleges or public databases to increase the diversity of the training data. If the researcher includes images from different sources he must normalize all images to decrease the variation (Moen et al., 2019). Another way is data augmentation operations like adding flipped images, images with different zooming, or rotation to the training data (Moen et al., 2019).

Considering the workload to generate proper training data, it is tempting to use pre-trained networks. Unfortunately, pre-trained networks have several risks and they are likely to do more damage than good (problem of pre-trained networks are discussed later).

Overall, the researcher is encouraged to generate his own training data for his experiments. Fortunately, there are multiple software for image processing and generating training data like for example, ImageJ is a free open-source software for image processing and parts of the generating can be automated via macros. (Schindelin et al., 2015).

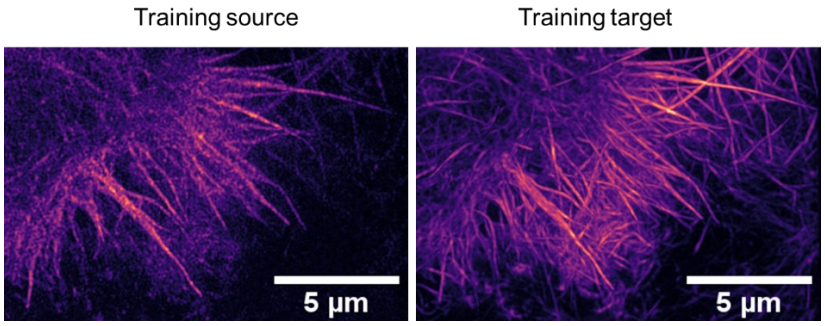


Figure 5. The training data contains paired images. CARE includes the low-quality image (training source) and corresponding high-quality ground truth image (training target). Images above are patches and they are enhanced to demonstrate the difference in images.

1.4.2 Training the network

After the training data is either generated or chosen, the next step is the training of CNN (in this section referred to as “network”). The training for the network consumes time but allows a new unseen input data processed without training. This means that multiple sets of unseen images can be processed if their quality remains the same. This feature is useful because image processing and analyzing are faster with a pretrained network compared to manual image processing. To train networks the three things are required: Access to powerful GPU, specific softwares and programming skills.

Access to powerful GPU: Deep learning requires millions of parameters (Moen et al., 2019) and a high level of accuracy (Carneiro et al,. 2018) to execute properly. Not only it stresses a regular computer a simple execution would take a long time. For example, in the medical field time is an important factor when the patient is diagnosed; delays may cost lives (Carneiro et al., 2018). For researchers, time is also a crucial resource. Then regular computers contain a central processing unit (CPU) which fits poorly to the execution of deep learning. Graphics processing units (GPU) or Tensor processing units (TPU) perform much faster than regular CPUs but they are costly and often not a

default option in regular computers. The solution is to either buy and assemble it yourself or perform cloud computing where the user uploads the training data to the cloud and GPU is provided by a company.

Softwares and programming: Assembling deep learning needs specific software and programming skills. Building the CNNs requires programming and Python is a common programming language but also R and C++ are used (Angermueller et al., 2016).

Fortunately, the need for customization of deep learning for biological science has been recognized in computer vision communities (Belthangady and Royer, 2019). The pipeline of deep learning has been modified and the development aims now to more interactive graphical tools which help to explain how deep learning works, how the researcher can control the process, and how results are achieved (Belthangady and Royer, 2019). There are several commercial products to perform deep learning today. Some products have a plugin to software like DeepImageJ for Image or Fiji to perform deep learning with pretrained networks and CPU. The plugin is easy to assemble and use in the software but it does not allow the user to train his own network. Moreover, assembled deep learning softwares like TensorFlow and PyTorch require the use of GPU to be efficient and ordinary computers contain only CPU which makes these softwares impractical and slow. Some products provide cloud-based platforms for serious deep learning research like Amazon SageMaker but they are costly.

All these tools have their strengths and weaknesses and are suitable for certain situations (possibility to invest time or money, or large amounts of GPU power or disk space are needed). For the researcher who is new to deep learning and wishes to use it with minimum costs and is not familiar with programming the software should have following properties: First, it demonstrates how deep learning works and expresses clearly the actions needed to properly perform. The user without mathematical or

technical background should be able to use the platform without enormous prestudies. Also, the platform should allow the modification of deep learning methods via simple programming. Second, it provides the free GPU power and disk memory to perform the training of the network and procession of the unseen images. For the large-scale deep learning usage this may not be possible but for the free introduction to deep learning as the researcher does not have to invest in hardware and assembling. Thirdly, the platform should allow us to save and share results and trained networks to colleagues or the public without extra work. Based on this the deep learning platform Google Colaboratory (Colab) was chosen.

Google Colab is a free web-based platform provided by Google LCC. It contains the computing power for deep learning and utilizes Google Drive cloud systems. In this environment, notebooks are easily performed, edited, and shared. It utilizes Jupyter notebooks and they are easily distributed in GitHub. The downside of being free has limited resources that affect the training parameters mentioned early. The most troubling components are the use of deep learning framework Tensorflow, the time limit, and the memory limit.

Tensorflow is an open-source library developed by Google Brain Team (Abadi et al., 2015) and it was released in 2017. It provides a collection of tools to perform deep-learning and deep-learning methods. The supported programming languages are Python, JavaScript, and Swift. All DL methods used in the thesis (StarDist, N2V, and CARE) were supported by Tensorflow 1.0. Tensorflow 2.0 was released in 2019 (TensorFlow Team. 2019). Unfortunately, Tensorflow 2.0 does not support StarDist and might be unsupportable for N2V and CARE. The problem is well acknowledged and Tensorflow 1 is still available in Google Colab. Unfortunately, this is a temporary solution because Tensorflow 1 is no longer updated and it will be unusable in the future.

Google Colab provides a free environment for users with small-scale resources and research. Because of this, the notebooks used in the Google Colab environment can be continuously executed up to 12 hours (Colaboratory - Frequently Asked Questions, 2020). The time limit prevents the user from monopolizing all resources available in the cloud. The time limit enables the maximum number of users at the same time.

Three relevant factors affect the time limit. The first factor is the decision between processing units GPU developed by Nvidia and TPU developed by Google Cloud. Free version of Google Colab often utilizes four different GPU sources: Nvidia K80s, T4s, P4s and P100s and the user can use one of them at the time. The user cannot choose which GPU source to connect his Colab and this means that the user cannot utilize the fastest GPU source (Colaboratory - Frequently Asked Questions, 2020). Google Colab has not revealed the exact RAM limits and amount of RAM memory varies time to time. Generally, the 12 Gbit RAM limit is reported (Caneiro et al,. 2018). Google Colab Pro was released in 2020 which allows the user to utilize the faster GPU but it is charged.

Google Colab utilizes either the GPU (Graphics Processing Unit) developed by Nvidia or TPU developed by Google Cloud. Using either of these processing units allows the execution of the notebook without consuming the user's CPU (central processing unit) resources.

The second factor is the training time of the network. The number of epochs defines how many times the network is trained to improve its denoising ability. The number of steps is quite irrelevant concerning the training time but it might affect the validation error. The third factor is the regular runout of the RAM which occurs after a few minutes when the operating browser tab is not active. This means that the user is required to keep the browser tab activated to keep executing the notebook. Fortunately, the runout creates the checkpoints, and re-activating

the browser tab allows the restart of performing from the previous checkpoint.

Two memory limits affect the usability of Google Colab. Google Colab provides approximately free 12 GB RAM for performing notebooks. However, this resource is not guaranteed. Google actively monitors the usage of the users' RAM resource and prioritizes RAM for users who have used resources less recently. Thus, it is advisable to close all unnecessary notebooks. (Colaboratory - Frequently Asked Questions, 2020).

When datasets are large (especially the 3D microscopy images), the memory limit is exceeded quickly. Fortunately, the training data can be split into patches. Patches contain three parameters: size (length x width), height in pixels, and the number of patches per image (Figure 6). The size and height values must be equal or smaller than image values. For example, for the image size 1024 x 1024 (containing 33 slices), the maximum batch height must be less than 32 (divisible to 8) and the batch size must be less than 1024 (divisible to 8). Patches should not be confused with batches. Batches are the number of patches which are seen in each step during the training. Sometimes batches are defined as several images loaded to RAM runtime.

The second limit is associated with the disk space. Google Colab provides a few sample datasets for the user. In this folder (60 GB), the user can load his datasets. Unfortunately, this folder offers temporary space for datasets, and all files are deleted after closing the notebook or the regular runout. The temporary folder is another way to regulate the usage of common memory resources.

To increase the space needed for the training dataset, the user can mount his Google Drive Account to Google Colab. Google Drive provides free 15 GT storage and thus the user can use larger datasets. Moreover, the mounting allows the saving of the results directly into Google Drive, and datasets are not deleted after the notebook is closed. Unfortunately, high-quality microscopy images (especially 3D movies) are large files. This

may be an issue during the training of the network and storing the results. Fortunately, users can purchase extra storage space and prices are quite fair.

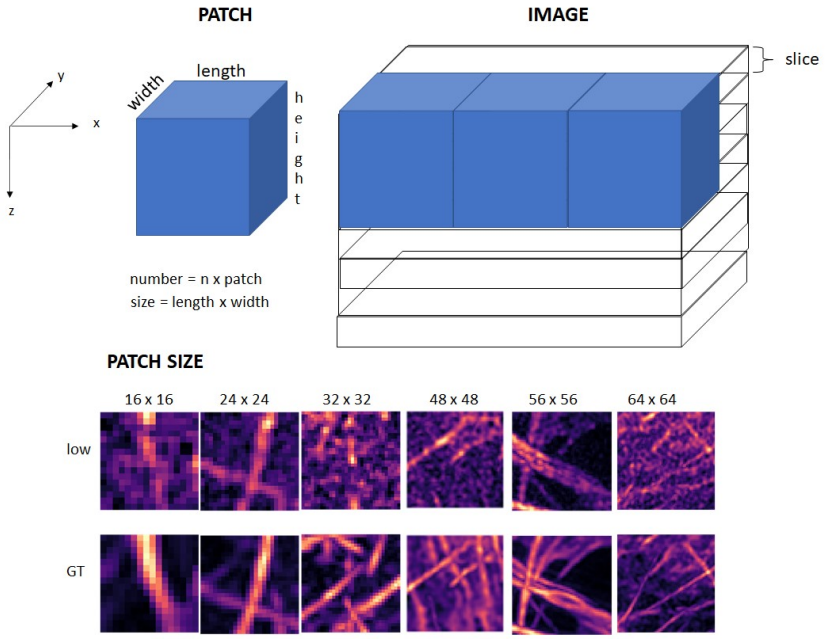


Figure 6: Parameters of patches: size, height, and the number of patches. The patches below were used for CARE (3D) optimization. The first row contains the low SNR images and the second row contains the corresponding ground truth images.

1.4.3 Evaluation of trained model

Network training must be properly done. Failing to do so, the network predicts images poorly and generates artifacts and hallucinations. These three components to remember about training the network. The first thing is to understand that the trained network performs well only if they are used on data that are similar to the data used to train the networks. If the pretrained network is not used correctly, it will lead to mistakes and the generation of artifacts.

Second, two main components result in inappropriate training: the quality of the training dataset (discussed in the previous paragraph) and training parameters. If they are not chosen or adjusted properly the trained network results in overfitting or underfitting. Overfitted network refers to the situation when the variance between the training and samples are high (Figure 7). Ovefitted networks have not learned the general way to handle images and this causes problems when the unseen images are introduced to the overfitted network. Overfitted model results often from too simple training data or too many epochs. Underfitted network is the opposite of overfitting and it often refers to the situation where the training data is so complex that the network does not find the general way to handle them. (Moen et al., 2019).

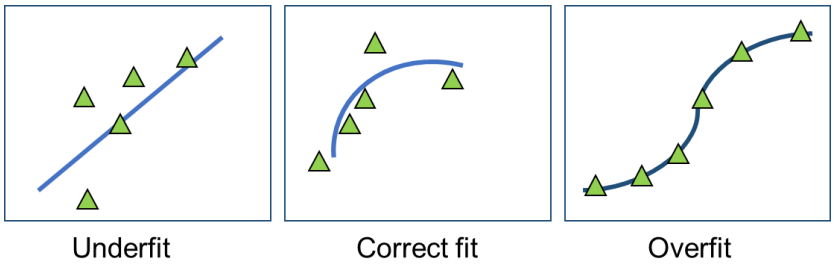


Figure 7. Learning curve must be not be underfit nor overfit. Blue line describes the network learning and green triangles real-life samples. Underfitting (left) results when the network performs too simple results compared to real-life samples. Overfitting (right) results when the network learns samples too well and does not find the trend in real samples. A trained network with the correct fit (middle) finds good balance and can predict unseen samples.

Estimation of overfitting or underfitting can be studied by training error (loss error) and validation error (loss error). In optimal training, the validation error should be slightly higher than the training error. This represents the situation where training data

are complex enough to reach the same values as the validation error. Moreover, validation is challenging enough to inhibit overfitting (Figure 8).

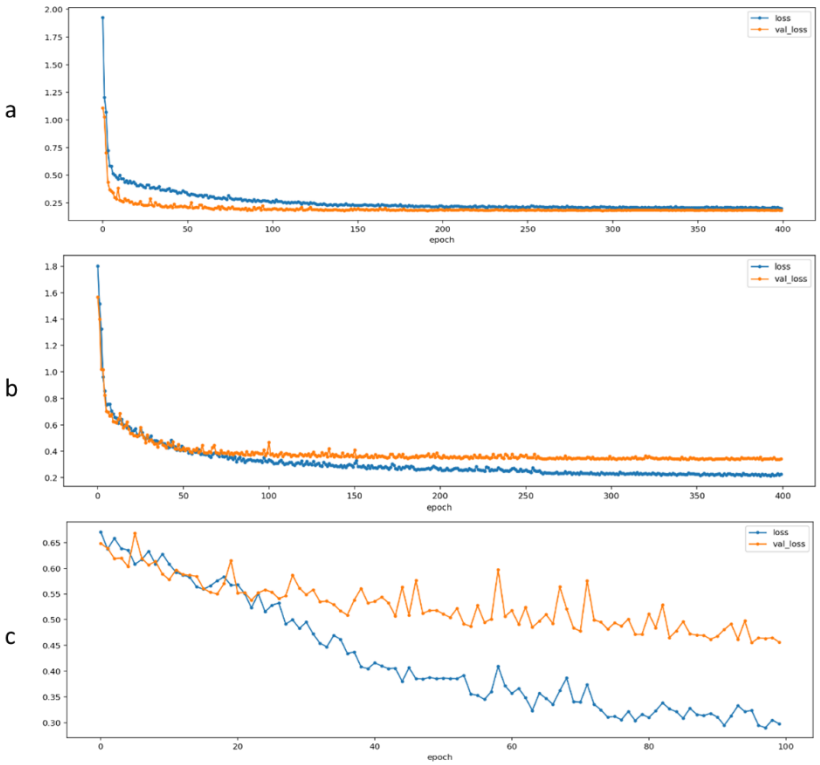


Figure 8. Estimation of the trained network. Loss function (blue) describes the error during the training from training images and the loss function of validation images (orange) describes the error from the validation images. The a-figure represents well-balanced training data and training parameters. The b-figure represents the overfitting problem when the training data are too simple and the c-figure reveals the overfitting problem when training data are too complicated.

Inappropriate training not only results in poor images but it also may cause the network to recognize patterns that are not there (hallucination problem, see Figure 9). For example, when a human is looking at the clouds and sees the shapes of animals (Belthangady and Royer, 2019). Another reason for hallucinations is the challenging background of the training images.

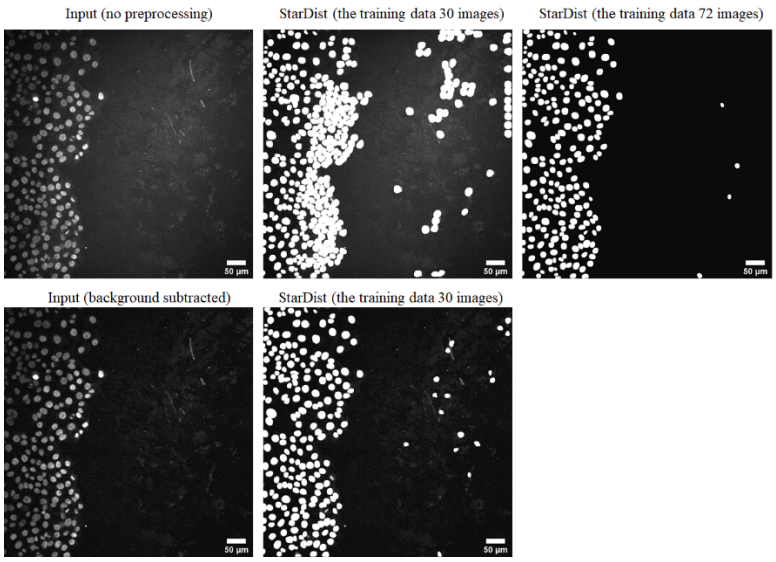


Figure 9. Inappropriate training data or training the network results in unreliable segmentation. In this case the challenging background in the input image caused the extra objects on the edges and in the frontline of the cells. The problem may be resolved by increasing the training data (upper row) or preprocessing the input image (lower row). The input image in the upper row is from DCIS.COM (ZeroCostDL4Mic) training data. The lower was not included in further study. Ideally the preprocessing is not needed.

Third, training the network (and preparing the training data for training) consumes RAM memory and disk memory resources (see cultural barriers above). This set limits the training parameters (discussed below) and the researcher must often adjust the training parameters of deep models to the resources available.

1.4.4 Prediction

When the network is trained properly it is time to introduce to the unseen data. Unlike the preparation of the training data or training itself, the prediction phase is easy and fast. The same

pretrained network can be used in multiple sets of the unseen data if their quality remains the same.

Prediction should be designed that images are easy for further processing. For example cell ROIs provided by StarDist should be easy to use in ImageJ for further processing. Results should also contain the trained model (and all parameters), quality control results and predictions as in one package (folder). One crucial principle is that predictions are new images (for example when images are denoised) and original images remain unmodified.

2. Aims

This Master's thesis is part of project ZeroCostDL4Mic. The purpose of the project is to provide free tools for training and implementation of deep learning (Unet, fnet, StarDist, N2V, and CARE) to microscopy images in Google Colab platform for researchers with no previous experience of deep learning (Chamier et al., 2020). The project is a collaboration between two research groups: Docent Guillaume Jacquemet from Cell Migration Lab (Åbo Akademi University, Turku, Finland) and Prof. Ricardo Henriques from HenriquesLab (UCL, London, England).

In this thesis, I will generate user-friendly Google Colaboratory notebooks that can be used by non-experts to train their networks online and for free (Scientific question 1). The notebook that I will create can be used to denoise microscopy images (Scientific question 2) and for the segmentation of cell nuclei images (Scientific question 3). I will use these notebooks to restore and analyze my videos, but the final version of notebooks will be tested by users to estimate their usability.

In this master's thesis, we concentrate on three methods: Content-aware Image restoration (CARE) and Noise2VOID (N2V) for noise removal and StarDist for segmentation of cell nucleus (object detection) from microscopy images.

The ultimate goal of the thesis is to make deep-learning tools more accessible for users with no technical background to use, learn, and share these methods in the field of microscopy image analysis. The goal is split into three parts and they are explained in the following chapters.

2.1 Objective 1- The user-friendly workflow

Implementing deep learning from scratch requires a technical understanding of computer hardware, software, and coding.

Deep learning requires heavy calculation power and thus high-cost processors, powerful RAM (virtual memory), and graphic cards are essential. Users with no technical background might find this to be an unnecessary burden compared to traditional ways of analyzing images. Deep-learning approaches also begin to be available through commercially available software, but the license price can rarely be met by individual laboratories.

Google Colab is a free website provided by Google LCC. The site contains a cloud-computing system and computing power needed to perform deep learning. Google Colab has some limitations concerning RAM and timeout, but it is a useful option for a user who is interested in deep learning but has no experience of programming or deep learning.

Implementing Google Colab includes several challenges. Uploading the data to Colab might be slow and impractical and storage space of Google Drive is limited to 15 GB. This is an issue when microscopy images – and datasets – are large files. Computing power is needed especially for training networks and, thus, the RAM memory limit is an issue when restoring 3D images. Runtime limit is 12h in Google Colab and this limits the number of epochs (rounds) in training networks.

Objective: To create user-friendly deep learning notebooks (object detection and image denoise) performed in the Google Colab platform. Notebooks are created according to the literature (N2V: Krull et al., 2018, CARE: Schmidt et al., 2018 and StarDist: Schmidt et al., 2018). To achieve this, the main focuses are:

User-friendly interface: Generate an interface through which a user can learn how deep learning works. The interface explains various processes and images to illustrate the process. The data are automatically downloaded and provided.

The control of the process: Users can train their network (patches and network) and use the network on one of their images (validation). A process folder is also created including highlighted options (for example, process images in ImageJ/Fiji).

Sharing the framework: Users can share their Google Colab notebooks to showcase their research and framework for future developers. For that, the structure of the notebook is coded as stable as possible.

2.3 Objective 2 - Object detection by StarDist

Cell migration is studied in multiple ways and research would benefit from the creation of automated tracking strategies. Object detection is a useful method for the detection of cells and cell nuclei from microscopy images. The StarDist method utilizes the star-convex polygon to detect nuclei boundaries via CNNs. (Schmidt et al., 2018)

Segmentation is a crucial step in image processing because the analysis is based on what segmentation extracts from the background. Another challenge is how segmented clusters of cell nuclei are separated by the StarDist network. Expected results include proper segmentation cells from the background. After separation, all possible cell nuclei lumps are separated properly to achieve a reliable result.

Objective: The aim is to generate a pipeline to train StarDist to automatically track and analyze video of migrating cells. The pipeline is created by combining original notebooks via coding and emphasizing the interface. Once the analysis pipeline is established, the research group uses it to analyze their cell migration movies.

2.2 Objective 3 - CARE versus N2V

The research group has an interest in studying the role of cellular protrusions during cancer cell invasions and, especially, the role of filopodia at cell-cell junctions. The dynamics of filopodia between cells are imaged using structured illumination microscopy (SIM). Unfortunately, imaging is delicate due to bleaching and phototoxicity. Combining long-term live imaging acquisition with deep learning would enable longer imaging.

Objective: To find optimal parameters to run CARE using Google Colab within the limitations (see section *1.4*). After that, CARE abilities are compared to Noise2VOID to restore live imaging data in Google Colab. Once the optimal parameters are found, these networks are used to train and restore the research group's images.

3. Materials and Methods

3.1 Object detection

The training data for StarDist contains paired images: the original microscopy image and corresponding mask image (see the example in Figure 10). Two different StarDist training data were used (Table 1). Original training data (DSB2018, Schmidt et al. ,2018) and prepared (ZeroCostDL4Mic (StarDist), Jukkala and Jacquemet. 2020).

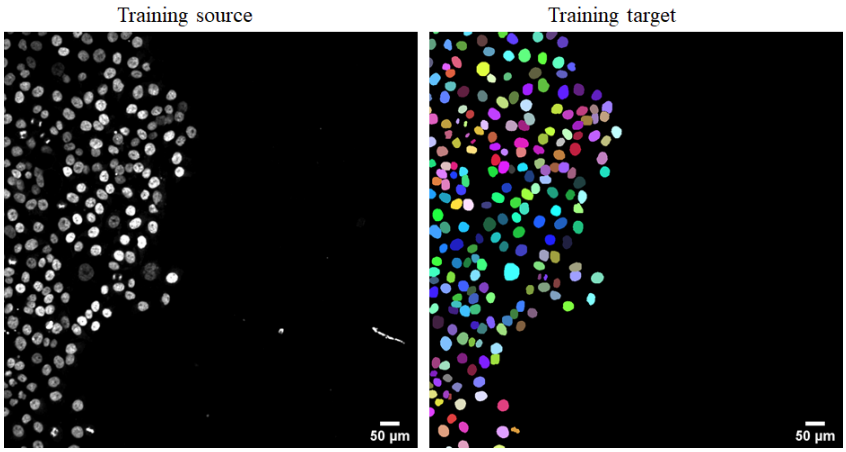


Figure 10: Supervised training data contains paired images from which algorithm studies and learns. Paired training images (original and mask image respectively) from the DCIS.DOM training dataset (Jukkala and Jacquemet, 2020).

ZeroCostDL4Mic (StarDist) training data (Jukkala and Jacquemet, 2020) was generated for this Master thesis. Paired images were generated in Fiji (Schindelin et al., 2012, procedure in Appendix 1). The image was duplicated and the segmentation to the background and the foreground was done by adjusting the threshold of the image. After segmentation, the particles were calculated from duplicated images. All artifacts and partial cells were removed. The ROI was drawn for every remaining unmarked cell and finally, all cells in the image had ROI. Finally,

ROIs were labeled and the duplicated image was renamed the same as the original image. The dataset is freely available. DSB2018 training data (Schmidt et al., 2018) is the subset of the stage1_train images from the Kaggle 2018 Data Science Bowl (Ljosa et al., 2012). The dataset is freely available.

Table 1. Training datasets for the 2D StarDist method.

	DCIS.COM (ZeroCostDL4Mic)	DSB2018
Data type	72 paired microscopy images (fluorescence) and corresponding masks	447 paired microscopy images (fluorescence) and corresponding masks
Microscopy data type	Fluorescence microscopy (SiR-DNA) and mask (manual segmentation)	Fluorescence microscopy images and masks
Microscope	Spinning disk confocal microscope with a 20x 0.8 NA objective	Diverse modalities
Cell type	DCIS.COM LifeAct-RFP cells	Several cell types and stains
File format	.tif (16-bit for fluorescence and 8 and 16-bit for the mask images)	.tif (8-bit for training images and 16-bit for the mask images)
Image size	1024x1024 (Pixel size: 634 nm)	multiple
Author(s)	J. Jukkala, G. Jacquemet. 2020	Schmidt et al. 2018 Original data: V. Ljosa, K. L. Sokolnicki & A. E Carpenter (2012).

3.2 Noise2Void

The training dataset for denoising images by N2V includes one low SNR image. The self-supervised neural network studies the low SNR image and denoise images.

Two different training datasets (Table 2) were used for 2D N2V. U-251 dataset (A. Stubb et al., 2020) contains 2D microscopy images of U-251 glioma cells (paxillin-GFP tagged). A2780

dataset (Jacquemet, 2017) contains 3D microscopy images of A2780 ovarian carcinoma cells, transiently expressing Lifeact-RFP. As ideally, one needs only a single noisy image for self-supervised N2V, there are no GT images. Both datasets are freely available.

Table 2: Training datasets for N2V (2D and 3D)

	U-251 (ZeroCostDL4Mic)	A2780 (ZeroCostDL4Mic)
Data type	Microscopy images (fluorescence)	3D microscopy images (fluorescence)
Microscopy data type	Fluorescence microscopy (paxillin-GFP)	Fluorescence microscopy (Lifeact-RFP)
Microscope	Spinning disk confocal microscope with a 63x 1.4 NA objective	Spinning disk confocal microscope with a 63x 1.4 NA objective
Cell type	U-251 glioma cells, endogenously expressing paxillin-GFP	A2780 ovarian carcinoma cells, transiently expressing Lifeact-RFP
File format	.tif (16-bit)	.tif (16-bit)
Image size	512x512 (Pixel size: 248 nm)	512x512x13 (Pixel size: x,y: 195 nm , z: 500 nm)
Author(s)	A. Stubb *et al.* 2020	G. Jacquemet (2017)

3.3 CARE

The training dataset for denoising images by CARE includes several paired images: original image (low SNR images and corresponding ground truth images (GT) of the original. A neuron network studies the low SNR image and compares it to the GT image and denoise images.

Images used for prediction were either stacked 2D or 3D images containing variable background (Table 3). All images contained challenging conditions such as uneven illumination, image noise, and saturation.

Table 3: Training datasets for CARE (2D and 3D)

	SIM.DCIS.2D (ZeroCostDL4Mic)	SIM.DCIS.3D (ZeroCostDL4Mic)
Data type	21 paired microscopy images (fluorescence, 2D) of low and high signal-to-noise ratio	20 paired microscopy images (fluorescence, 3D) of low and high signal-to-noise ratio
Microscopy data type	Fluorescence microscopy (Lifeact-RFP)	Fluorescence microscopy (Lifeact-RFP)
Microscope	Structured Illumination Microscopy (SIM) with a 60x 1.42 NA objective	Structured Illumination Microscopy (SIM) with a 60x 1.42 NA objective
Cell type	DCIS.COM Lifeact-RFP	DCIS.COM Lifeact-RFP
File format	.tif (32-bit)	.tif (32-bit)
Image size	1024x1024 (Pixel size: 40 nm)	1024x1024x33 (Pixel size: x,y: 40 nm, z: 125 nm)
Author(s)	G. Jacquemet (2020)	G. Jacquemet (2020)

3.4. Notebooks

All DL methods are performed in Jupyter Notebooks (Jupyter, 2020). Jupyter is open-source software for interactive computing containing executing, developing and executing code and sharing results (Jupyter, 2015). All notebooks were programmed by Python 3.7. Python is a programming language created by The Python Software Foundation.

Google Colab is a free web-based platform provided by Google LCC. It contains limited computing power (GPU, TPU) for deep learning and utilizes Google Drive cloud systems. In this environment, Jupyter notebooks are easily performed, edited, and shared. (Colaboratory, 2020)

Our ZeroCostDL4Mic Colab notebooks were made possible thanks to the instructions provided by the authors on how to implement their DL networks (CARE: Weigert et al., 2018,

Noise2VOID: Krull et al., 2019, and StarDist: Schmidt et al., 2018). We rewrote the code and generated a new user-friendly workflow optimized for Google Colab. For all methods, there were three original notebooks for data preparation, network training, and prediction of unseen images.

4. Results

This Master's thesis resulted six new notebooks: StarDist (2D and 3D), CARE (2D and 3D), and N2V (2D and 3D). These notebooks contain a new workflow to achieve user friendly interface (Objection 1). StarDist notebooks perform image segmentation for 2D and 3D microscopy images (Objection 2). CARE notebook performs image restoration and N2V performs denoising image and these two methods were compared to (Objection 3).

4.1. Objective 1 - The interface of the notebook

The goal of the thesis is to introduce deep learning for new users and thus the interface of the notebook must be simple to use but informative. The crucial point was to find a balance between simplicity and demonstration. The user with no previous experience of deep learning must learn the basics to operate the notebook. Still learning and operating the notebook must be easy and all obstacles which may worsen the user experience must be removed. These obstacles may be unnecessary technical information, visible code (the code is still easily accessible if the user wishes to see and learn how it works) odd error messages (which does not affect the execution), or just unsuitable workflow which causes the crashing of the notebook.

The interface was improved by the user feedback and workflow was improved by the creators of each method. The default workflow for StarDist, N2V, and CARE was done by Johanna Jukkala, Lucas von Chamier, Christoph Spahn, Guillaume Jacquemet and Romain Laine. The development of the notebooks continues beyond this thesis.

Based on the feedback the following guidelines for the interface were chosen: The notebook must be simple to use. All unnecessary information (technical and visual code) is hidden

behind the interface and revealed if needed. All operations which do not require any inputs for the user are combined into one cell (for example patch formation and generation of the default model). The number of actions is minimized.

The notebook must demonstrate how deep learning is executed. The notebook was divided into clear sections and info for all parts was provided. The outputs of the executions (images, values, figures) were shown. If the notebook is shared the new users must be able to read and understand the results easily. These sections are:

Introduction: This section (Figure 11) describes the purpose of this notebook, references (the original article and authors), and the original code from authors. The chapter also includes the creators of this notebook but highlights that the notebook is the combination of the original notebooks and inspired by the original code. The user is recommended to create an experiment folder on Google Drive. This allows the easy file upload from Google Drive to an external hard drive or other location. The premade experiment tree helps the user to understand the input options in the following chapters. Finally, as traditional bioscience training does not contain information technology studies, this chapter describes what is the notebook text cell, code cell, and how to execute and modify the notebook.

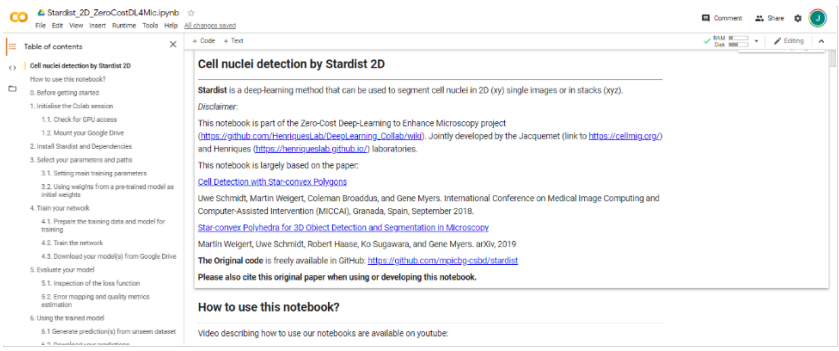


Figure 11. The general view of the StarDist notebook. On the right is table of contents. The latest version can be found in GitHub (Chamier et al., ZeroCostDL4Mic. 2020).

Preparations: Before the actual use of the method the notebook must be prepared (Figure 12). First, the user selects the processing unit (most often GPU) as it does not always default in Colab. After that, the user mounts his Google Drive to the notebook. Because the mounting allows access to all folders in the user's account, the user should mount only the trusted notebooks. Finally, the notebook requires certain libraries and modules to perform the notebook. The chapter does not require any input for the user, but it is advisable to specify each library in the code. This chapter may also contain the cell which enables Tensorflow 1. If TensorFlow 1 is upgraded in the future, it may need an extra library and objects.

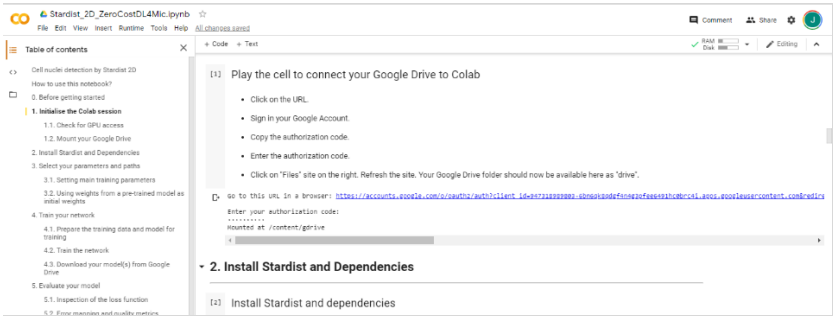


Figure 12. Mounting Google Drive and installing dependencies. Screenshot from the StarDist 2D notebook. The latest version can be found in GitHub (Chamier et al., ZeroCostDL4Mic. 2020).

Selecting parameters and path to the folders: When the notebook is ready the user can input training parameters and file paths to the training dataset (Figure 13). Depending on the method there are several inputs and if the user is unsure how to input, the default parameters are provided. The execution of the cell prints the size and resolution of images and output a few sample images from the training images.

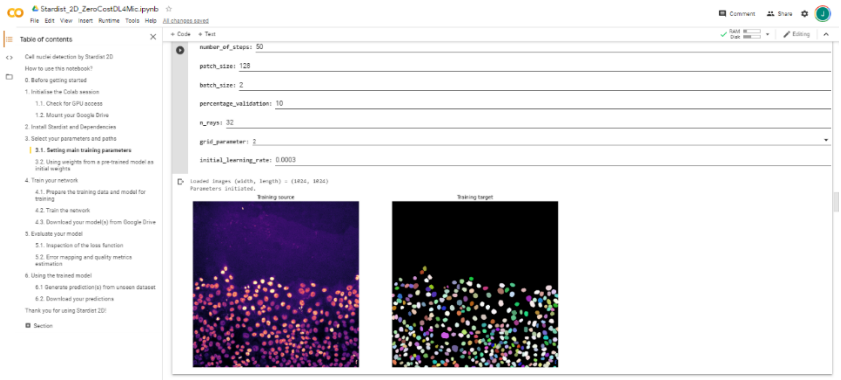


Figure 13. The sample images from the training images. Preview images not only visualize them but also informs that training data is succefully loaded. Screenshot from the StarDist 2D notebook. The latest version can be found in GitHub (Chamier et al., ZeroCostDL4Mic. 2020).

Data preparations and model setup:

In this section, training images are generated to the training dataset. Images are divided into patches and later splitted to training patches and validation patches according to the parameters input in above. The general rule is to use 10% patches for validation. The original training images remain unmodified and the training dataset is the separative object. Also, the default model is generated. The output contains a training dataset and a default model (Figure 14). Few examples of patches are printed to ensure proper data preparation. This section requires no special actions from the user.

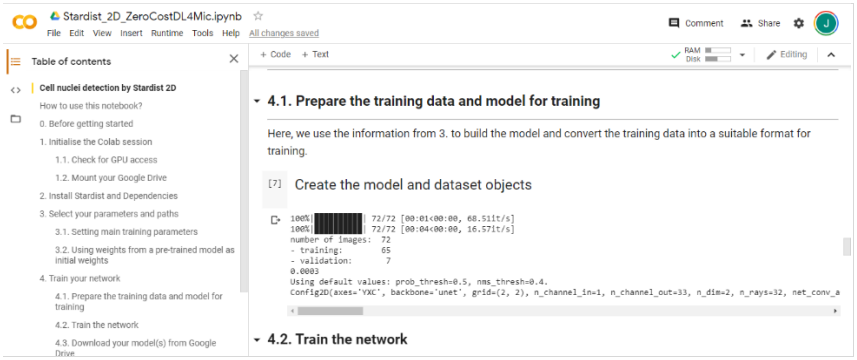


Figure 14. The training dataset and default model is generated in one cell as they do not require any inputs. Screenshot from the StarDist 2D notebook. The latest version can be found in GitHub (Chamier et al., ZeroCostDL4Mic. 2020).

Training the network: Training the network is usually the most time-consuming part of the method and takes from minutes to hours. The training process is shown and the intermediate results per epoch are printed (Figure 15). During the training, the results are saved to the model. This section requires no special actions from the user. The training dataset remains unmodified and the trained model is saved into the results folder.

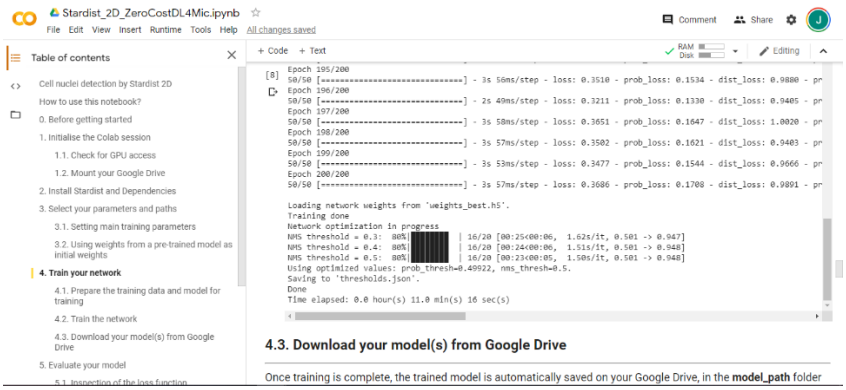


Figure 15. The training results are printed and saved to the results folder. Screenshot from the StarDist 2D notebook. The latest version can be found in GitHub (Chamier et al., ZeroCostDL4Mic. 2020).

Evaluation of the training: The quality of the trained network is estimated by comparing the training error and validation error during the training (Figure 16). The learning curves are presented in the figure for the estimation. If the user is not content with the results, the network can be retrained by modifying the training parameters. The user can use a pretrained model if there is any.

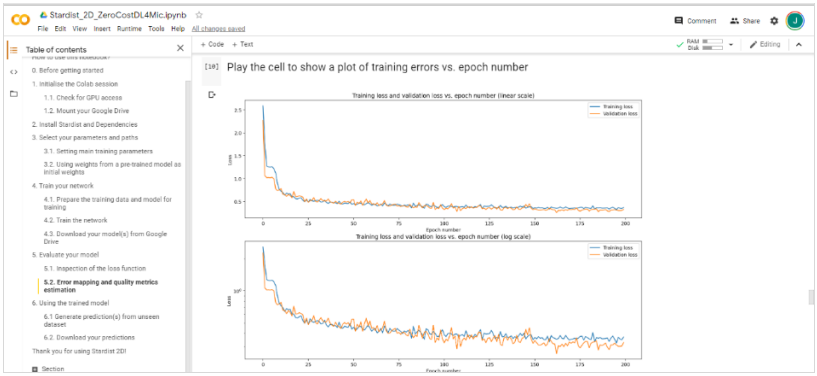


Figure 16. The comparison of loss error and validation error. Screenshot from the StarDist 2D notebook. The latest version can be found in Github (Chamier et al., ZeroCostDL4Mic. 2020).

The quality control (QC) tests the trained model on the few unseen images. For the evaluation of StarDist the intersection over union (IOU) is used for CARE and N2V the Structural Similarity Index (SSIM), Root Squared Error (RSE) were used and added to N2V, Peak signal-to-noise ratio (PSNR) was used (Figure 17 and 18). The guidelines that these values mean and what is desirable were explained for the users who are not familiar with these metrics. The results were visualized as they demonstrate the difference between the resulting image and ground truth image. This helps the user spot challenging parts of the image and may help improve the image quality in the future. The implementation of QC was not done by Johanna Jukkala.

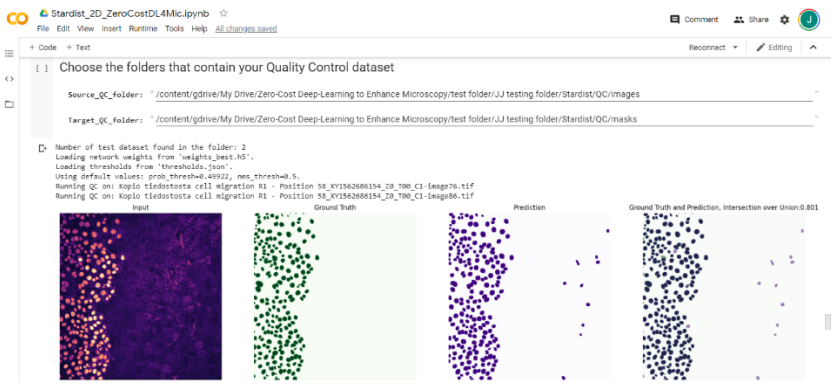


Figure 17. The quality control for StarDist method. All images (input, ground truth, prediction) are easily copied for presentation. Screenshot from the StarDist 2D notebook. The implementation of QC was not done by Johanna Jukkala. The latest version can be found in GitHub (Chamier et al., ZeroCostDL4Mic. 2020).

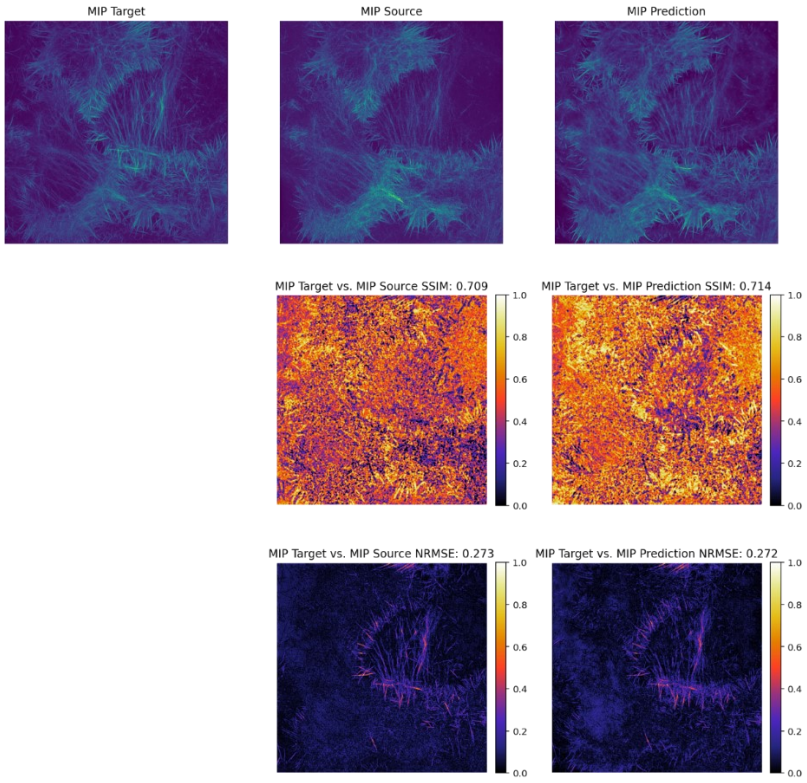


Figure 18. The quality control for CARE method is visualized to help user to understand the metrics and presenting results user-friendly. Upper row contains raw images of target, source and prediction image. Second row demonstrate visually the difference between target ja prediction images via SSIM and on

the lowest row same thing but via NMRSE. Screenshot from the CARE 3D notebook. The implementation of QC was not done by Johanna Jukkala. The latest version can be found in Github (Chamier et al., ZeroCostDL4Mic. 2020).

Unseen data and saving the results: In the final chapter, the trained network is used to predict unseen images (Figure 19). The sample results are printed to ensure proper execution. Finally, all results (images, ROIs) are saved into Google Drive. The original unseen images remain unmodified.

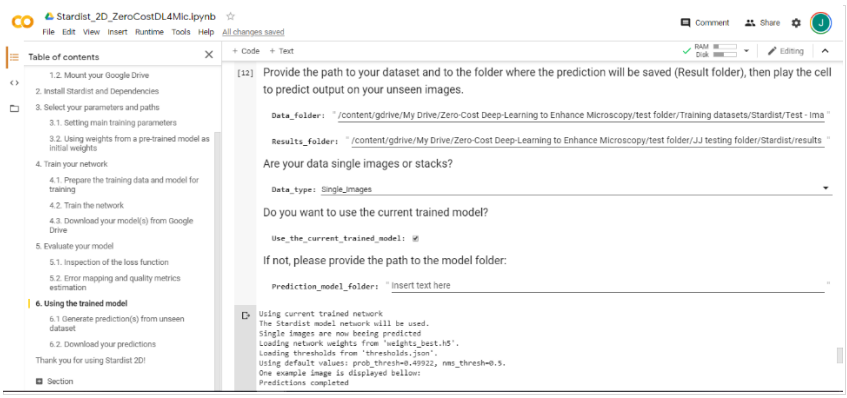


Figure 19. The trained network is executed and the unseen images are used. Screenshot from the StarDist 2D notebook. The latest version can be found in GitHub (Chamier et al., ZeroCostDL4Mic. 2020).

The original code is modified to ensure fluent execution of the notebook and then it is notated as the inspiration of the original code. Because some users might find the code visually intimidating the actual code was hidden behind the interface. If the user is interested in modifying the notebook, the code can be revealed by double-clicking the cell.

Sometimes the notebook crashes unexpectedly. To ease the troubleshooting the code cells are programmed to print results of each code cell. For example, the proper creation of batches is ensured by printing a few batch images. Failing to do so, the cell

prints error messages and possible repair advice are printed. Overall, the development of all workflows was continued beyond this thesis.

4.2. Objective 2 - Object segmentation: the effect of the training data

Object 2 resulted in two StarDist notebooks for image segmentation (2D and 3D microscopy images). The optimal parameters for two training data were studied. Overall, the quality of the training data affects greatly the training results.

Google Colab provides limited memory resources and it has a time limit. This may be challenging if the user does not understand how to optimize the training process. Thus, the effect of the processing unit (GPU and TPU), patch size, and the number of steps and epochs were studied.

To prevent the overfitted model the user can improve training results by increasing the patch size. Unfortunately, it consumes Colab resources. Thus, the effect of patch size was studied for both GPU and TPU. As expected, larger patch size improves the trained model (until it starts to overfit) and results in better detection (Figure 20).

Overall, GPU performs better than TPU as the TPU reaches the time limit much sooner than GPU. As mentioned earlier, TPU is recommended for long-term training and our studies seem to agree with this. TPU consumes less memory capacity than GPU but the time limit in Google Colab becomes the problem. Though one must remember that training data is quite small (only 72 images) and TPU may perform better with larger training data and if there is no time limit. Moreover, the memory resource is not quarantined and thus the actual runout may happen earlier. This means that the memory usage should be below 10 Gbit to prevent the runout.

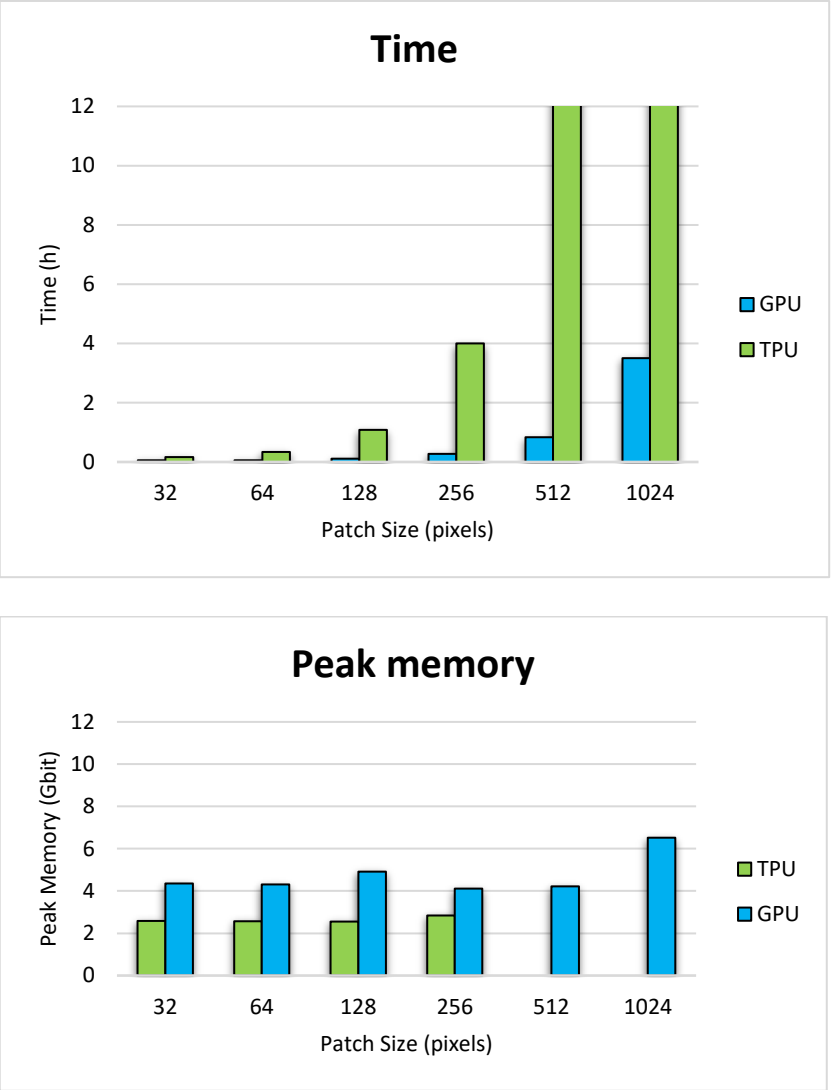


Figure 20. TPU process larger data better than GPU but due to time limit and memory limit in Colab, GPU is more effective than TPU. Time limit (the upper image) is exceeded when TPU (green) was used and it is not suitable to use in these restrictions. GPU (blue) consumes more memory capacity than TPU but stays below the 10 Gbit (the lower image). The training parameters were the same in all studies: 72 training images (10% for the validation) and 50 steps/200 epochs. The trained model was overfitted when the patch size was 32. The training time was calculated based on ms/steps as the time limit was exceeded for TPU (patch size 512 and 1024) and memory consumption was not recorded.

Overall increasing the patch size improved IoU value (1.0 is perfect). The IoU was slightly better for GPU than TPU (Figure 21). The reason for this remains unknown, but again TPU is meant to train large training datasets. Nevertheless, it is safe to say that GPU is the best option for Google Colab.

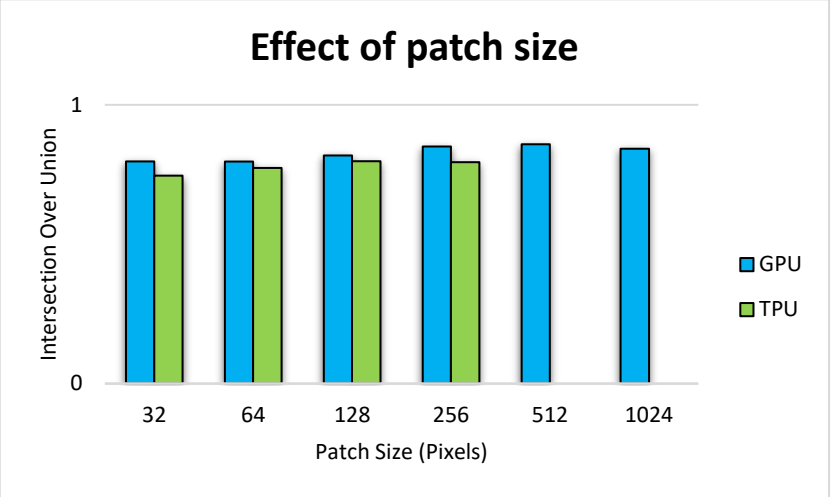


Figure 21. Increasing the patch size and using GPU (blue) results in the best IoU values but only slightly improvement over TPU (green). P values for each patch sizes (32, 64, 128 and 256) were 0.0038, 0.7721, 0.8965, and 0.3341 respectively. The training parameters were the same in all studies: 72 training images (10% for the validation) and 50 steps/200 epochs. The trained model was overfitted when the patch size was 32.

The effect of steps and epochs were studied. The patch size of 256 pixels was chosen as it has shown good IoU in Figure 22. The number of epochs increases the training time but improves IoU values. For this training dataset 50 steps and 200 epochs were optimal parameters to train the model.

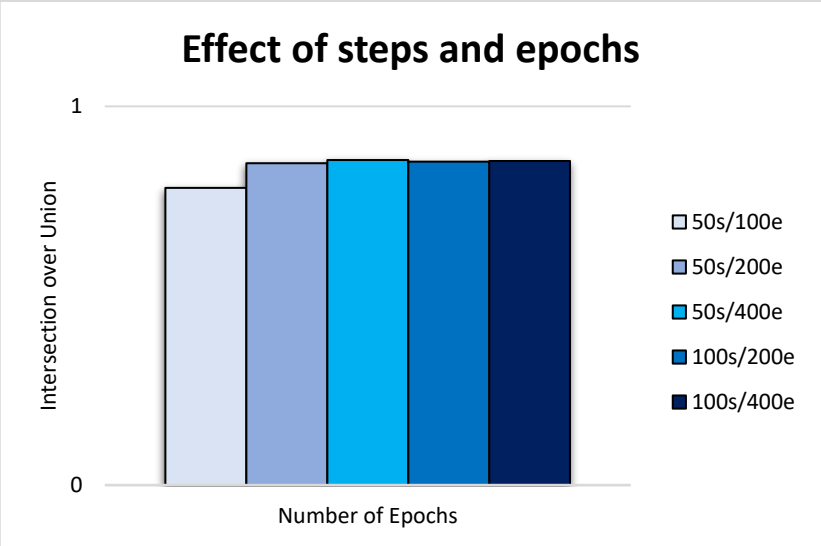


Figure 22. IoU varies little above 50 steps and 200 epochs. The training parameters were the same in all studies: 72 training images (10% for the validation) and patch size 256 pixels. GPU was also used in all studies.

Finally, the parameters were tested on the original StarDist training data (DBS2018, see Table 1). The trained model detected cells poorly when the DBS2018 dataset was used for the training (Figure 23 and Figure 24). There may be several reasons for this. DBS2018 contains 447 images that have different resolutions, sizes, and magnification. This may make training data too complicated for our images and the trained model makes mistakes. Our DCIS.COM contains 72 images with the same resolution, size, and magnification and are similar to the unseen images. This makes it suitable and this is shown as better IoU values. However, it is important to understand that this study does not label DSB2018 as poor training data. It is just not suitable for this study.

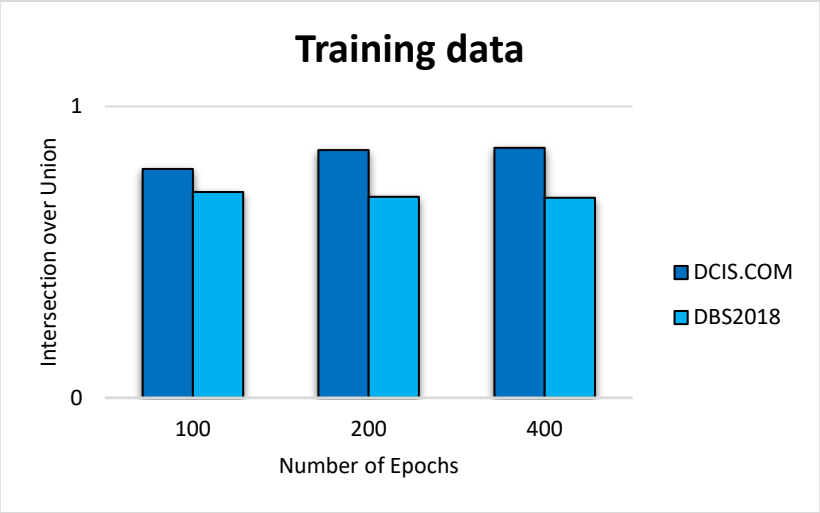


Figure 23. The training data affects the training quality. DCIS.COM (dark blue) training data results in a better model. DSB2018 (light blue) training data may be too complicated for the study as it contains a large variety of cell images. For all sets (numbert of epochs: 100, 200 and 400) P value was less than 0.05. The training parameters were the same in all studies: 72 training images (10% for the validation), patch size 256 pixels and 50 steps, and 200 epochs. N= 13, data is shown as average plus minus SD.

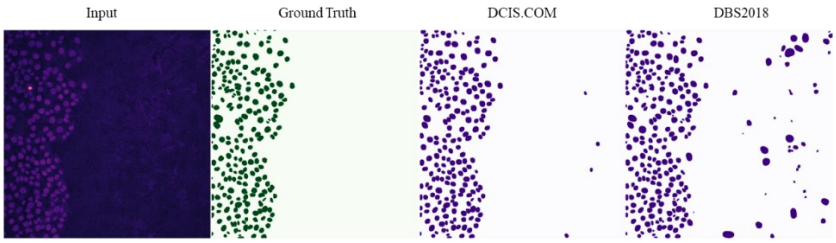


Figure 24. Unsuitable training data may result poor trained model. The model trained on DBS2018 (far right) makes several mistakes compared to DCIS.COM (middle right) when input image (far left) was used. IoU was for DCIS.COM and DBS2018 training datasets were 0.850 and 0.690 respectively. The training parameters were the same in all studies: 72 training images (10%

for the validation), patch size 256 pixels and 50 steps, and 200 epochs.

Overall, the selection of training data is a crucial step. Even if DSB2018 is the original training data for cell detection, it failed to train the proper model for our images. Therefore, the user is strongly encouraged to generate their own training data. Failing to do so the trained model makes mistakes. Not because the training data is poor but not suitable for the user's images. But if the user wishes to use premade training data the user should pay attention to the contained training data and decide is it suitable for his images.

The same awareness is necessary when pretrained models (other than own) are used. Training may take several hours and optimize may cause the runout of Google Colab resources, which consumes more time. As for using premade training data, the pretrained models are not unequivocally poor. Still, they are a riskier choice. They have not only unknown training abilities, but they have also been trained with unknown training data. This makes them even more risky choices than premade training data.

As for the conclusion, before the user starts training the model, the training data must be chosen by properties of the unseen images which the user plans to use in his research. The trained network performs well if the unseen images and training images are similar. To avoid poorly trained models, the user is encouraged to generate his own training data and train his own model instead of using the premade training data or pretrained models. Generating the training data consumes time but it must be done once. Also training the model takes time but after training the model consumes time but a well-trained model processes the unseen images quickly.

Moreover, in the Google Colab environment, the optimal use of StarDist for cell detection is to use GPU instead of TPU and using the largest patch size possible (within the GPU RAM). When

optimizing the training parameters the user may start with the small patch sizes and the low number of epochs to save memory resources and after that increase them to maximum to reach the optimal parameters for his training data.

4.3 Objective 3 - CARE versus N2V

Objection 3 resulted two CARE notebooks for image restoration and two N2V notebooks for denoising images (2D and 3D microscopy images). The optimal parameters for two training data were studied. Overall, the quality of the training data affects greatly to the training results. Patch size limits the use of notebooks as memory usage is increasing along the patch size. CARE performs slightly better than N2V when the training data contains complex images. Both methods are useful and suits for different situations.

CARE 3D

Patch size affects greatly to the training results generally the larger patch size improves the results. The effect of the patch size was studied for both training time and peak memory as they are the main restrictions in Colab. The training time is not a relevant issue if the number of epochs remains low (Figure 25). Memory usage is increased significantly as the patch size and number of patches are increased. To improve the training results the patch size should be increased but memory capacity is easily exceeded. As the memory capacity is not stable in Colab, this causes the main challenge for the training.

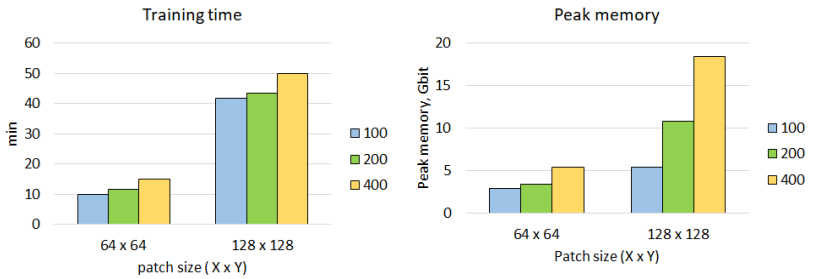


Figure 25. Training time increases little when patch size is higher. The real issue is memory usage and peak memory increases rapidly as patch size increases. The batch size (how many patches are introduced at the same time to the training) were 100 (blue), 200 (green) and 400 (yellow). The training parameters in all experiments were: batch size 16, 50 steps and 100 epochs.

The quality control shows little difference between the experiments even if it improves moderately the results (Figure 26). As all the models became easily overfitted it implies that the size of the training dataset is too small. At this point it was not possible to gain extra images and thus results remained the same. CARE improved the image quality (Figure 27) by removing noise but left the fine parts smudged (Figure 28).

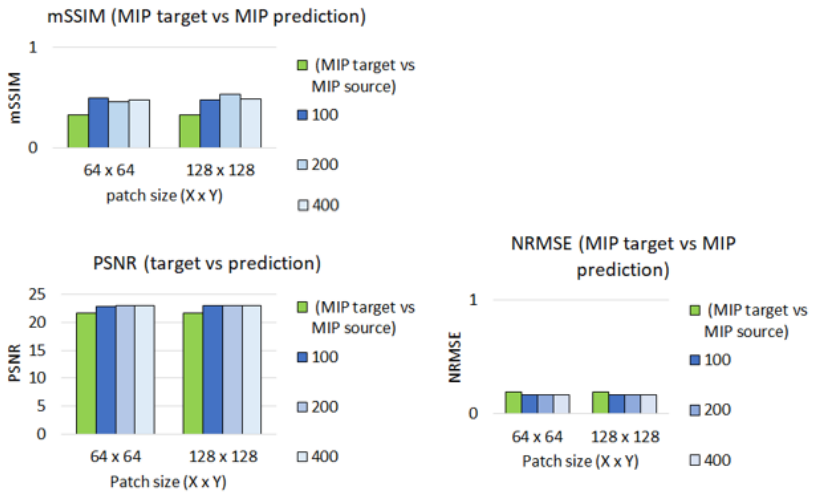


Figure 26. Variation between the quality control results is low and maybe caused by the small size of the used training data. P value

was not calculated due to lack of images. The training parameters in all experiments were: batch size 16, 50 steps and 100 epochs.

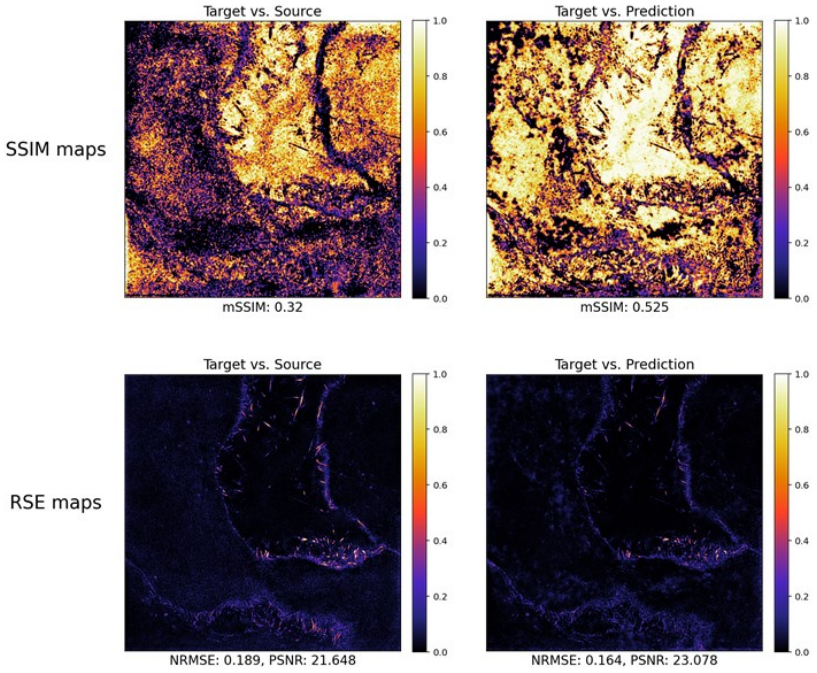


Figure 27. Detailed quality control of the one experiment (the blue bar in the Figure 26). Overall, the quality of the input image is improved in SSIM map (image becomes lighter in “Target vs Prediction” part) but not much in RSE map (image should be darker in “Target vs Prediction” part). The training parameters: patch size: 128 x 128 pixels, path height 8 pixels, number of patches per image: 200, number of patches total 4000, batch size 16, 50 steps and 100 epochs.

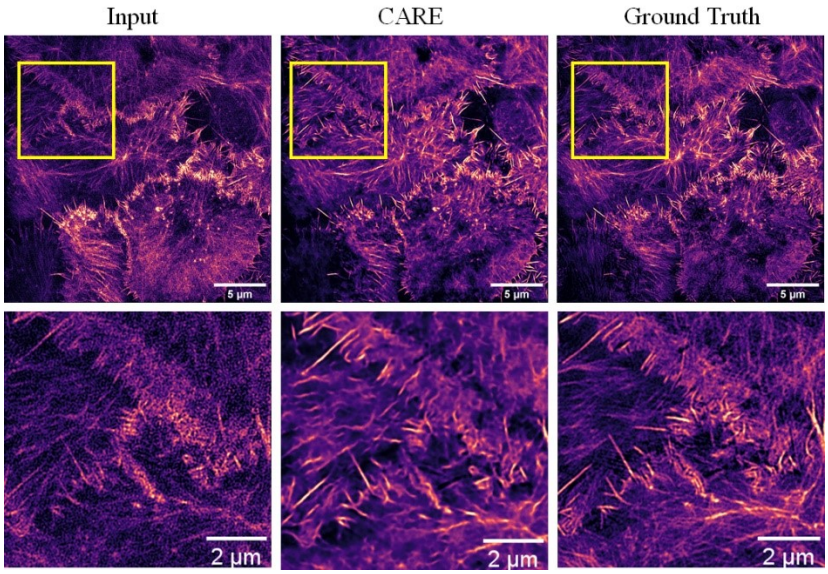


Figure 28. CARE (middle) improves the image quality from the input image (left) but smudges the fine parts of the image compared to the GT image (right). Reason for this may be small training data size. See the training parameters in Figure 26.

CARE 2D

The size of the training data can be increased by splitting the training data to patches. As previously is shown this may cause the exceed of the memory capacity of Colab. Another way to improve results is to create augmented training data where images are created by rotating and mirroring the images. In the Figure 29 show that again larger patch size affects little to the training time but greatly to the peak memory. Augmented training data consumed less the memory capacity and larger patch sizes were able to use.

Figure 29. Training time increases little when patch size is higher. The real issue is memory usage and peak memory increases rapidly as patch size increases. Augmented data (20 images augmented to 630 images, the green bar) consumes less memory than patches (20 images - > 4200 patches, the blue bar). Memory limit exceeded when patch size was 512 x 512 pixels and patches were used. The training parameters in all experiments were: batch size 16, 50 steps and 100 epochs.

Augmentation improves image quality only moderately even the patch size was able to be increased (Figure 30). The use of patches from augmented data was not possible as it exceeded the memory limit all the time. As in the case of 3D images CARE improved the image quality (Figure 31) by removing noise but left the fine parts smudged (Figure 32).

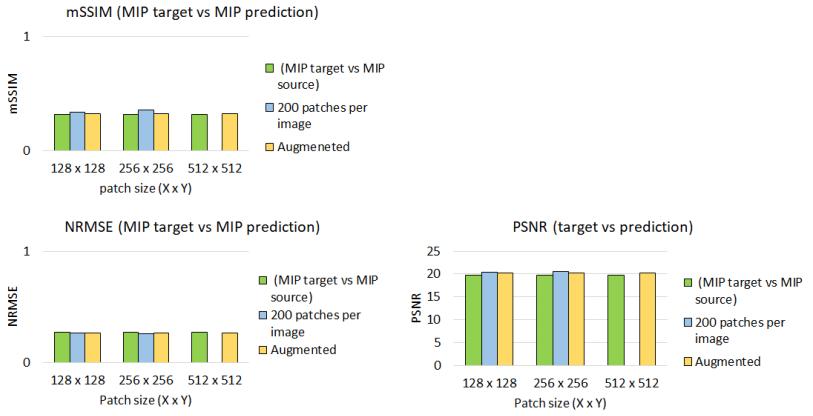


Figure 30. Variation between the quality control results (mSSIM, NRMSE and PSNR) is low and maybe caused by the small size of the used training data. Augmented training data allows to use of larger patch size but it does not improve the results. Augmented data (yellow) contained 20 images augmented to 630 images (n = 630) and 200 patches per image (blue) contained 20 images converted to 4200 patches (n=4200). Due to lack of images p values was not calculated. The training parameters in all experiments were: batch size 16, 50 steps and 100 epochs.

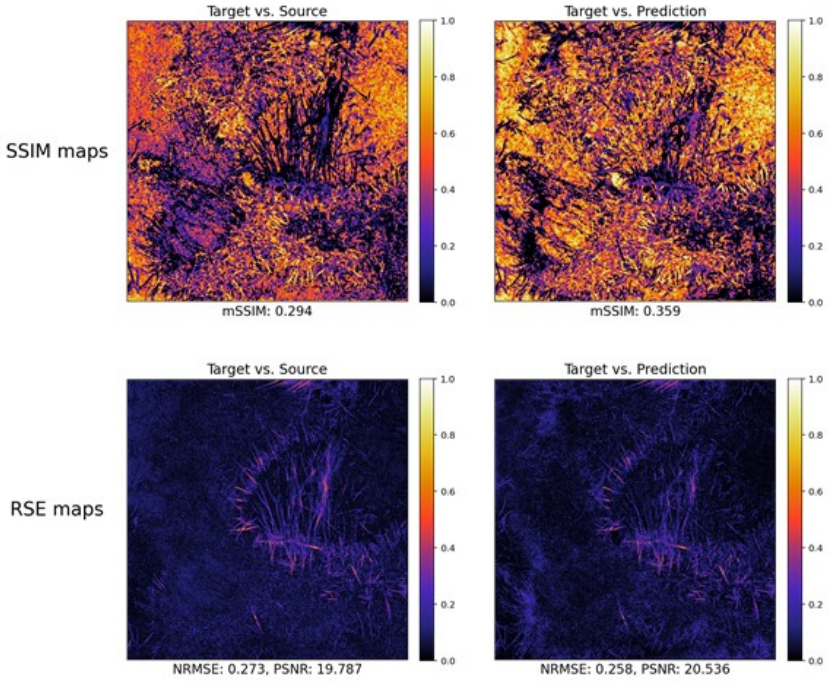


Figure 31. Detailed quality control of the one experiment (the blue bar in the Figure 30). The quality of the input image is improved but not much for both SSM or RSE maps. The training parameters: patch size: 256 x 256 pixels, path height 8 pixels, number of patches per image: 200, number of patches total 4200, batch size 16, 50 steps and 100 epochs.

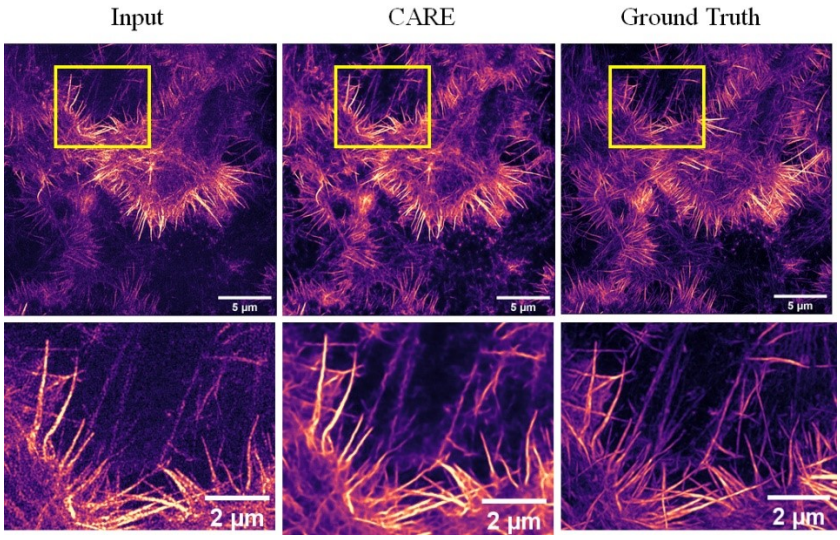


Figure 32. CARE (middle) improves the image quality from the input image (left) but smudges the fine parts of the image compared to the GT image (right). Reason for this may be small training data size. See the training parameters in Figure 30.

N2V 3D

The training data of N2V contains one microscopy image. Thus, the effect of the patch size was studied. In all studies, the training time and the peak memory were below the limits (Figure 33). Still there the studies had to be done a few times as the Google Colab may allocate resources differently and thus the training could take longer time or memory was runout. Unfortunately, the trained model became easily overfitted (Figure 34).

A. Krull mentions that N2V struggles to denoise high-frequency details like isolated bright pixels. The training data did not contain these kinds of areas but N2V performed well on the bright areas. These areas are surrounded by lower frequency pixels and thus the gap between the bright area and the environment was not large.

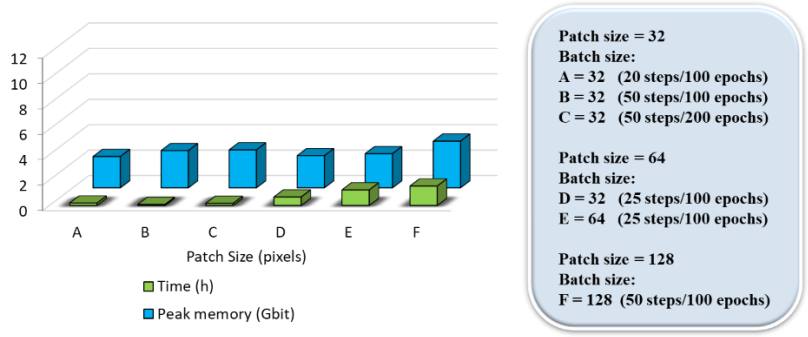


Figure 33. The training time (green) and the peak memory (blue) were below the time and memory limits but the model (A, C, and F) became overfitted quite easily.

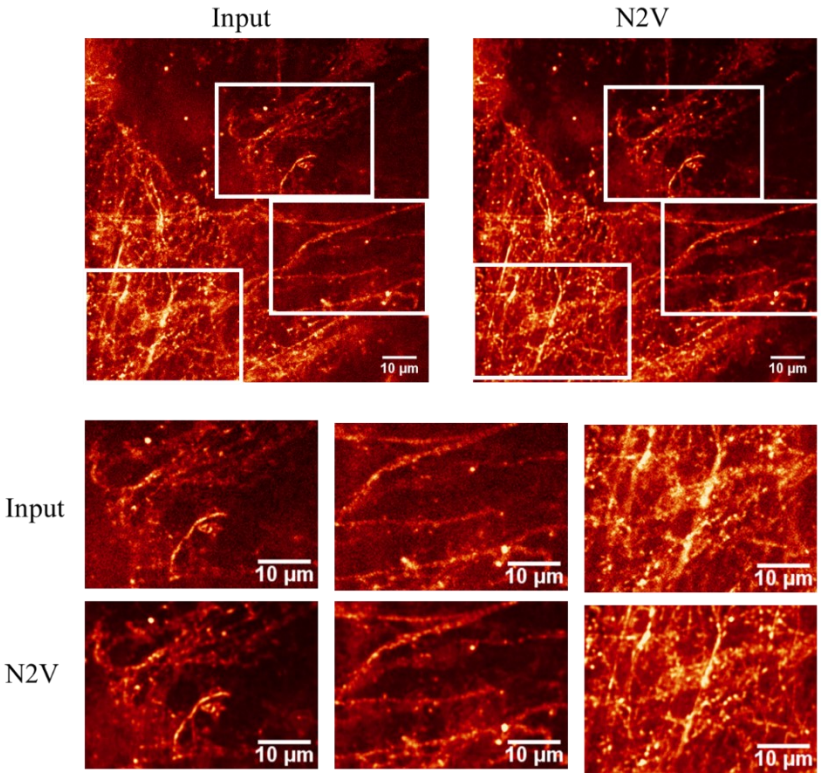


Figure 34. Images from the study E (Figure 33) N2V removes noise from the image but seems to smudge thicker noise areas. The bright areas are shown as normal. The training parameters: 25 steps, 100 epochs, patch size 64 x 64 pixels (total 1536 patches) number of batches 64.

N2V 2D

The effect of patch size and batch size (number of patches that are processed at the same time) were studied. The time limit and memory limit were not an issue but the learning curve showed that this model becomes overfitted quite easily (Figure 35). Still, the model denoise images nicely (Figure 36). Quality control was not done due to a lack of ground truth images.

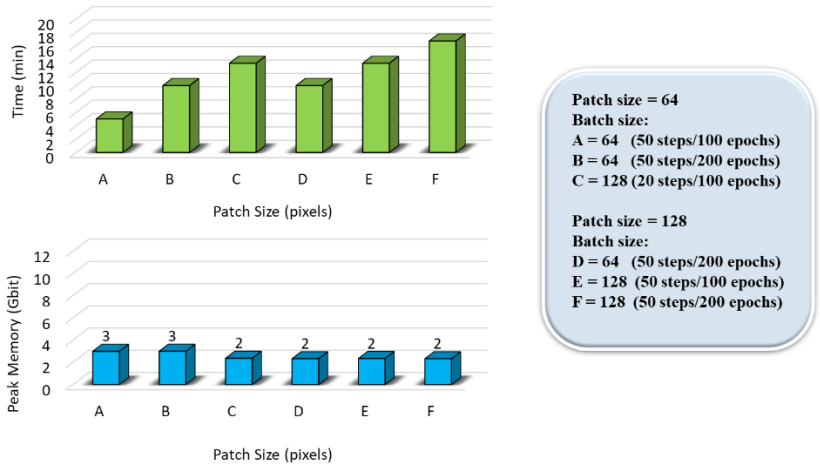


Figure 35. The training time (green) and the peak memory (blue) were below the limits but the model (D-F) became overfitted quite easily.

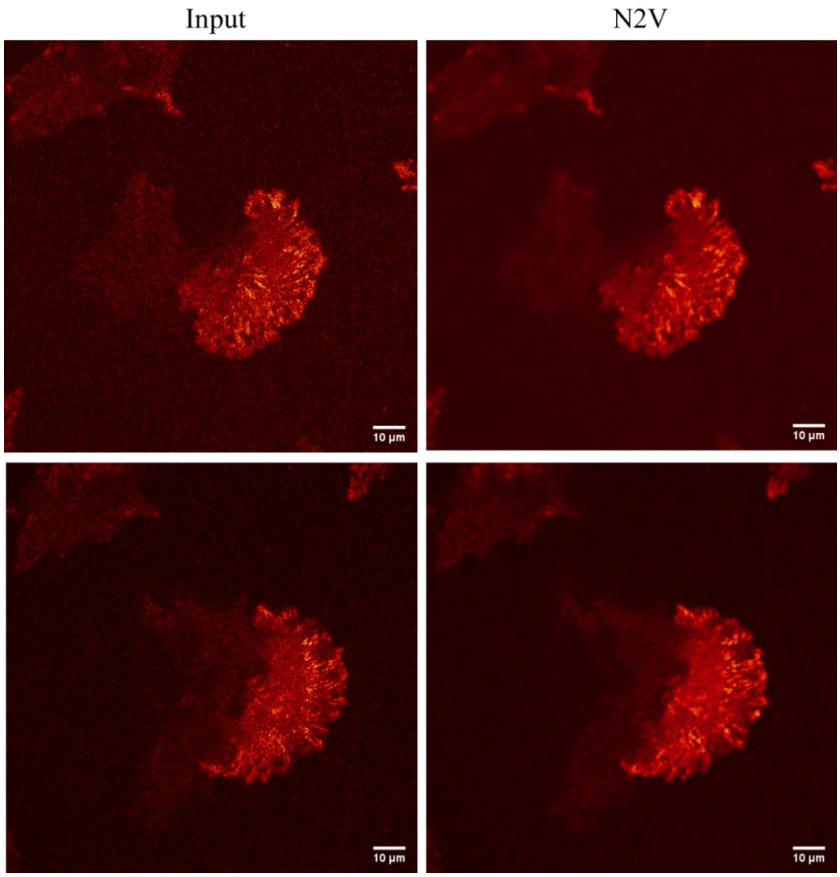


Figure 36. Unseen images from the B- study and N2V. The brightness/contrast values in denoised images were equalized to the input images to demonstrate the difference. The training parameters: 50 steps, 200 epochs, patch size 64 x 64 pixels (total 512 patches) number of batches 64.

N2V versus CARE

Finally, CARE was compared to N2V. The ideal training data would have been from CARE studies. Unfortunately, the training data from CARE studies was not sufficient as the honeycomb pattern in the images (caused by SIM imaging acquisition) prevents the use of N2V. The training data from StarDist studies contains masked images that can be used for CARE.

CARE improves images better than N2V (Figure 37). Both SSIM and NRMSE are very good compared to the input image. Altogether both CARE and N2V improve the images.

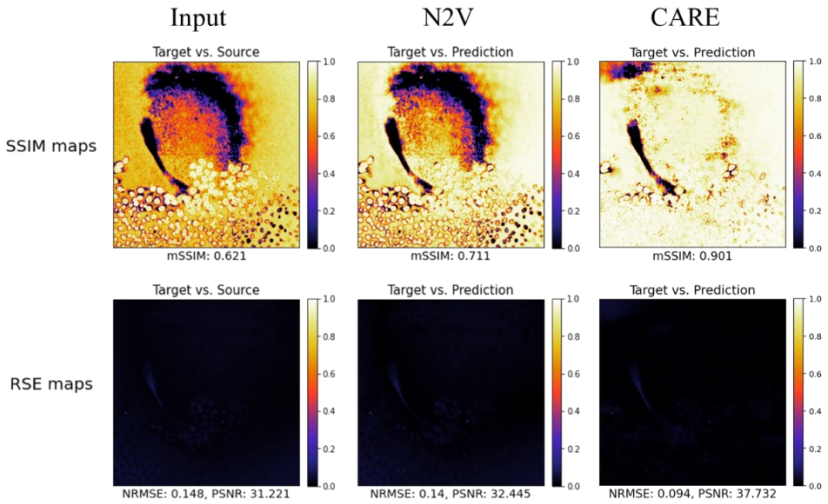


Figure 37. CARE improves the input image better than N2V for both mSSIM and NRMSE. The quality control of E-study (above, middle). The training parameters were for CARE: patch size 128, number of patches total 6000, 20 steps and 200 epochs and for N2V: patch size 128 number of patches total 6000, 50 steps, and 100 epochs.

In conclusion, CARE results generally better images but the training requires large training data. 3D images especially become the problem as the memory limit of Google Colab exceeds easily. Thus, increasing the training data to avoid the overfitted model is challenging and the use of CARE 3D is more limited in Colab than N2V. High-quality ground truth images may not be available for the researcher. The samples may be too fragile for long-time imaging or the number of images may be limited. Or the availability of the microscope may be the reason for the small number of images. In these cases, N2V is the only option and it can result in good results as shown above with 2D images. Based on these studies it seems that the N2V model becomes more easily overfitted than the CARE model. The reason for this may be that CARE is supervised and N2V self-supervised.

Still, it is not sensible to decide whether CARE or N2V is the method of choice. They both have challenges and advantages, and they should think of the tools for the proper situation. Also, if

the training data allows the can to be used as the comparative methods and this increases the value of the research. All this requires that the researcher understands the importance of the quality of the training data and how it behaves in certain parameters. Failing to do so, training results overfitted models and the model makes mistakes. Thus, the use of pretrained models and premade training data are risky as they are often designed for specific problems that may differ greatly.

For researchers who wish to use deep learning in his research, both CARE and N2V are a good method to work. The greatest work is to train the model but when done accordingly, the model processes the unseen images fast and intensifies the image process and analysis.

5. Discussion

Modern research contains complex data and processing is time consuming. Deep learning is one attempt to help researcher by analyzing tedious and time consuming data. Unfortunately, tools of deep learning may be expensive and requires often previous knowledge. Free tools for deep learning are a tempting idea when funding is limited or researcher wish to learn how deep learning works generally in his or her data.

The key components to create free deep learning tool are free access to GPU, availability to train own network, free storage and user-friendly program. The balancing of these aspects is inevitable and every deep learning tool has found their own. ImageJ has the free plugin DeepImageJ but it utilizes the user's CPU which is slow. DeepImageJ offers plenty of pretrained networks to use but pretrained networks are suitable only if the user knows how they are trained and what is used training data. Paid services like Amazon SageMaker offer vast resources for deep learning but require the user to learn a new program and invest in the product.

ZeroCostDL4Mic succeeds to be a free, open source and easily available tool. Free access to GPU is the most important asset as it is the costliest part in the deep learning process. Google Account and Google Drive are familiar for everyone and it makes distribution of results easy. ZeroCostDL4Mic is the open-source tool and it allows the user to see how deep learning is performed and the user can learn the programming behind the interface if the user is interested. The ability to train your own network allows the user to understand the process and what elements affect it. This is along the free access to GPU, one of the most important assets of ZeroCostDL4Mic. Prediction is the fast part of the process and proper training must be done only once.

Obviously ZeroCostDL4Mic has challenges as the other deep learning tools. Being the free tool means limited resources (time limit, memory peak) compared to the paid services. This

limitations inhibits the use of TPU at the moment. However, these challenges may be solved by optimizing the training properties to decrease the training time and need of GPU. This allows efficient small-scale use of deep learning for researchers in the biomedical field. Another challenge of ZeroCostDL4Mic is to become outdated. StarDist is performed by Tensorflow 1 which is to be replaced by Tensorflow 2. But currently StarDist does not run with Tensorflow 2 until it is reprogrammed by the original creators. But it is likely to happen as StarDist has proven to be useful to detect cells.

The difference between CARE and N2V favors CARE method but both are useful. If user can create proper training data CARE method is better choice but it amount of high quality images is scarce (needed as target images in training data) N2V the method of choice. User must also reserve images for statistics and in this thesis calculation of p values was not always possible.

The next steps for ZeroCostDL4Mic could be to introduce new image analysis aspects like add cell calculations (number, shape) to the StarDist and offer thus increase the options to use the results of object detection. In addition to CARE, N2V and StarDist the new image processing options could be added to ZeroCostDL4Mic. It is important to encourage the users to send constant feedback about bugs and possible suggestions for new tools to be included to ZeroCostDL4Mic. As the memory limit is the challenge in Colab the process should be improved to less memory consuming process as possible.

The thesis project was conducted and thesis written in 2020. The project was published in Nature Communications (Chamier et al., 2021) and the project and affiliated notebooks are found from: https://github.com/HenriquesLab/ZeroCostDL4Mic

6. Conclusion

Deep learning is a powerful tool for denoising images and detects cells from microscopy images. If properly used, deep learning allows us to process and analyze large datasets which would be normally time-consuming. The main challenges are unfamiliar technology for researchers, preparing the suitable training dataset and limitations of free resources.

Google Colab provides an easy and free way to introduce deep learning. Google Colab has limitations like memory limit and for massive usage of deep learning requires ultimately to invest in the computer hardware. Hopefully, the user-friendly platforms and notebooks (StarDist, CARE and N2V) encourage the researchers in the medical and biological areas to harness the power of computing and boost their research.

The Master's thesis is the part of project ZeroCostDL4Mic which was a collaboration of two research groups: Cell Migration research group (Docent Guillaume Jacquemet) at Åbo Akademi University and The Henriques lab (Professor Ricardo Henriques) at University College London in England. The project and affiliated notebooks are found from:
https://github.com/HenriquesLab/ZeroCostDL4Mic

7. Acknowledgments

I would like to thank Docent Guillaume Jacquemet for supervising this project and the responsible Professor Diana Toivola and Coordinators Raili Kronström and Joanna Pylvänäinen in Åbo Akademi University. I would like to thank Professor Ricardo Henriques, Dr. Romain Laine, and PhD Student Lucas Von Chamier from HenriguesLab for the collaboration project. I would like to thank Martina Lerche, Christoph Spahn, and Pieta Mattila for the testing of notebooks and valuable feedback. I would like to thank my family, friends and fellow students of the BIMA 2018 program for their support during the challenging time of the coronavirus epidemic in 2020.

8. References

Abadi M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen,, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, et al. 2015. Large-Scale Machine Learning on Heterogeneous Distributed Systems. https://arxiv.org/pdf/1603.04467.pdf

Angermueller, C., T. Pärnamaa, I. Parts, and O. Stegle. 2016. Deep learning for computational biology. *Mol Syst Biol*. 29;12(7):878.

Batson, J. and L. Royer. 2019. Noise2Self: Blind Denoising by Self-Supervision. *arXiv*:1901.11365 (Checked: 1.11.2019)

Belthangady, C., and L. A. Royer. 2019. Applications, promises, and pitfalls of deep learning for fluorescence image reconstruction. *Nature Methods.* 16:1215-1225.

Carneiro, T., R. V. Medeiros Da NóBrega, T. Nepomuceno, G. Bian, V. H. C. De Albuquerque, and P. P. R. Filho. 2018. Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning Applications. *IEEE Access*. 6:61677-61685.

von Chamier L., R.F. Laine, J. Jukkala, C. Spahn, D. Krentzel, E. Nehme, M. Lerche, S. Hernández-Pérez, P.K. Mattila, E. Karinou, et al. 2021. Democratising deep learning for microscopy with ZeroCostDL4Mic. *Nat Commun*. Apr 15;12(1):2276.

von Chamier, L., J. Jukkala, C. Spahn, M. Lerche, S. Hernández-Pérez, P. K. Mattila, E. Karinou, S. Holden, A. C. Solak., A. Krull, et al. 2020. ZeroCostDL4Mic: an open platform to simplify access and use of Deep-Learning in Microscopy. *bioRxiv* preprint
doi: https://www.biorxiv.org/content/10.1101/2020.03.20.000133v1

ZeroCostDL4Mic - Exploiting Google Colab to develop a free and open-source toolbox for Deep-Learning in microscopy. (GitHub) https://github.com/HenriquesLab/ZeroCostDL4Mic (Checked 01.04.2020)

von Chamier, L., R. F. Laine, and R. Henriques. 2019. Artificial intelligence for microscopy: what you should know. *Biochemical Society Transactions*. 47:1029–1040.

Chen, C. L, A. Mahjoubfar, L-C. Tai, I. K. Blaby, A. Huang, K. R. Niazi, and B. Jalali. 2016. Deep Learning in Label-free Cell Classification. *Nature - Scientific Reports*. 6:21471.

Colaboratory - *Frequently Asked Questions* https://research.google.com/colaboratory/faq.html (checked 04.02.2020)

Editorial policies. 2020. *Nature Research.* https://www.nature.com/nature-research/editorial-policies/reporting-standards (Checked 01.04.2020)

DeepImageJ. A user-friendly plugin to run deep learning models in ImageJ. 2020. https://deepimagej.github.io/deepimagej/ (Checked 27.6.2020)

Fang, L., F. Monroe, S. Weiser Novak, L. Kirk, C. R. Schiavon, S. B. Yu, T. Zhang, M. Wu, Kyle Kastner, et al. 2019. Deep Learning-Based Point-Scanning Super-Resolution Imaging. doi: https://doi.org/10.1101/740548

Google Cloud. *Cloud Tensor Processing Units (TPUs).* 2020. https://cloud.google.com/tpu/docs/tpus (Checked 10.04.2020)

Google Colab. *FAQ.* 2020. https://research.google.com/colaboratory/faq.html (Checked 25.6.2020)

Jacquemet, G. 2017. ZeroCostDL4Mic - Noise2Void (3D) example training and test dataset. https://zenodo.org/record/3713326#.Xsmt02gzbIV (Checked 10.05.2020)

Jupyter. The Jupyter Notebook - Introduction. 2015. https://jupyter-notebook.readthedocs.io/en/stable/notebook.html (Checked 10.05.2020)

Weigert, M., U. Schmidt, T. Boothe, A. Müller, A. Dibrov, A. Jain, B. Wilhelm, D. Schmidt, C. Broaddus, S. Culle, et al. 2018. Content-aware image restoration: pushing the limits of fluorescence microscopy. *Nature Methods.* volume 15:1090–1097.

Jukkala, J., and G. Jacquemet. 2020. ZeroCostDL4Mic - StarDist example training and test dataset. https://zenodo.org/record/3715492#.Xq8LZagzbIW (Checked 10.05.2020)

Krull, A., T-O. Buchholz, and F. Jug. 2019. Noise2Void - Learning Denoising from Single Noisy Images. arXiv:1811.10980. https://arxiv.org/pdf/1811.10980.pdf

Kumar, C. 2018 Artificial Intelligence: Definition, Types, Examples, Technologies. *Medium*. Aug 31.

https://medium.com/@chethankumargn/artificial-intelligence-definition-types-examplestechnologies-962ea75c7b9b (Web article, Checked: 31.10.2019)

Kraus, O.Z., B.T. Grys, J. Ba, Y. Chong, B.J. Frey, C. Boone, and B.J. Andrews. 2017. Automated analysis of high content microscopy data with deep learning. *Molecular Systems Biology*. 13:924.

Ljosa, V., K. L. Sokolnicki, and A. E. 2012. Carpenter. Annotated high-throughput microscopy image sets for validation. *Nature Methods*. 9:637.

.

Moen, E., D. Bannon, T. Kudo, W. Graf, M. Covert, and D. Van Valen. 2019. Deep learning for cellular image analysis. *Nature Methods.* 16:1233–1246.

Oppermann, A. 2019. What is Deep Learning and How does it work?. *Towards to Datascience*. Nov 12. https://towardsdatascience.com/what-is-deep-learning-and-how-does-it-work-2ce44bb692ac (Web article, Checked 01.04.2020)

Ounkomol, C., S. Seshamani, and M.M. Maleckar. 2018. Label-free prediction of three-dimensional fluorescence images from transmitted-light microscopy. *Nat Methods.* 15:917–920.

Richmond, R., A. Payne-Tobin Jost, T. Lambert, J. Waters, and H. Elliott. 2017. DeadNet: Identifying Phototoxicity from Label-free Microscopy Images of Cells using Deep ConvNets. arXiv:1701.06109

Ronneberger, O., P. Fischer, and T. Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. Medical Image Computing and Computer-Assisted Intervention (MICCAI). *Springer, LNCS*. 9351:234-241.

Salciccioli, J. D., Y. Crutain, M. Komorowski, and D. C. Marshall. 2016. Sensitivity Analysis and Model Validation. *Secondary Analysis of Electronic Health Records* | *Sensitivity*. Springer, Cham. pp 263-271.

Schindelin, J., C.T. Rueden, M.C. Hiner, and K. W. Eliceir. 2015. The ImageJ ecosystem: An open platform for biomedical image analysis. *Molecular Reproduction and Development.* 82:518–529.

Schindelin, J., I. Arganda-Carreras, E. Frise, V. Kaynig, M. Longair, T. Pietzsch, S. Preibisch, C. Rueden, S. Saalfeld, B. Schmid, et al. 2012. Fiji: an open-source platform for biological-image analysis. *Nat Methods.* 9:676–682.

Schmidt, U., M. Weigert, C. Broaddus, and G. Myers. 2018. Cell Detection with Star-convex Polygons. 2017. Lecture Notes in Computer Science book series (LNCS, volume 1071).

Stubb, A., G. Jacquemet, and J Ivaska. 2020. ZeroCostDL4Mic - Noise2Void (2D) example training and test dataset. https://zenodo.org/record/3713315#.XrjdJKgzbIV (Checked 10.05.2020)

Yang, W., X. Zhang, Y. Tian, W. Wang, and J-H. Xue. 2018. Deep Learning for Single Image Super-Resolution: A Brief Review. *arXiv*:1808.03344

Yao, K., Rochman, N. D., and Sun, S. X. 2019. Cell Type Classification and Unsupervised Morphological Phenotyping From Low-Resolution Images Using Deep Learning. *Scientific Reports*. 9:13467.

APPENDIX 1: Creating training dataset for StarDist in ImageJ

Original paper used for dataset training:

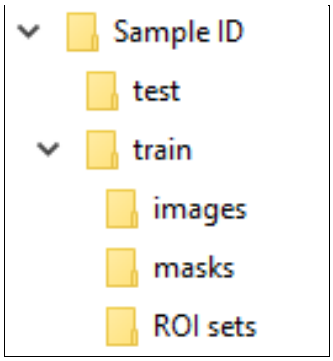
Uwe Schmidt, Martin Weigert, Coleman Broaddus, and Gene Myers.*Cell Detection with Star-convex Polygons*. International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI), Granada, Spain, September 2018. https://arxiv.org/abs/1806.03535

Martin Weigert, Uwe Schmidt, Robert Haase, Ko Sugawara, and Gene Myers. *Star-convex Polyhedra for 3D Object Detection and Segmentation in Microscopy*. arXiv, 2019. https://arxiv.org/abs/1908.03636

GitHub: https://github.com/mpicbg-csbd/StarDist

Steps to follow:

1. Create the following folder tree.



2. Select 20-40 images for training networks. Choose images that represent a general situation and save them into images-folder in tiff-format.

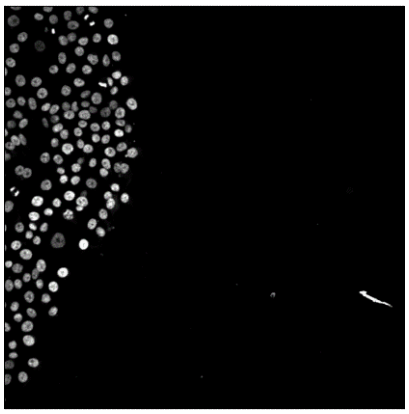


image 1.tiff

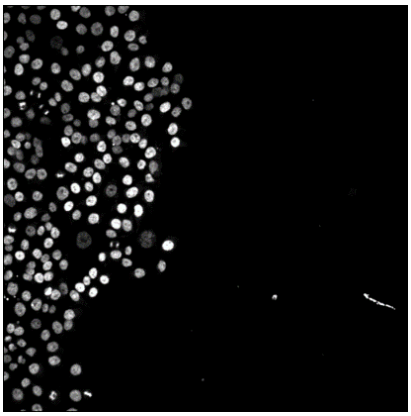
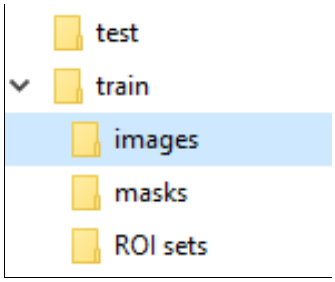
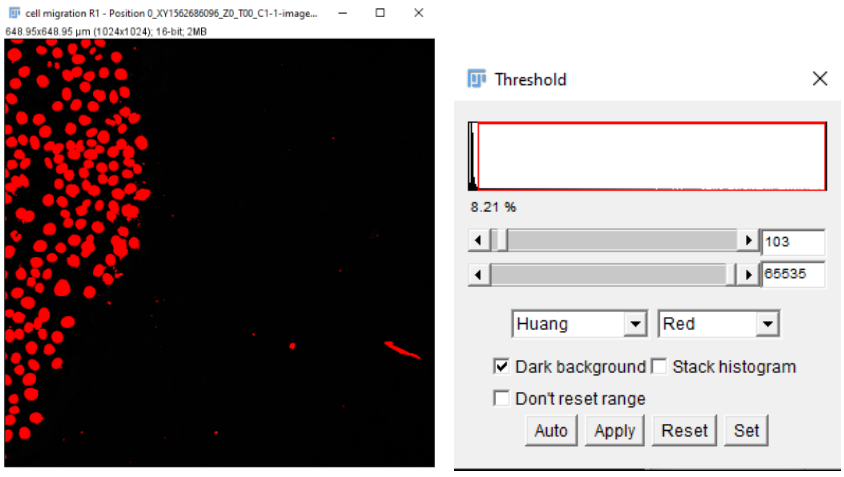


image 2.tiff

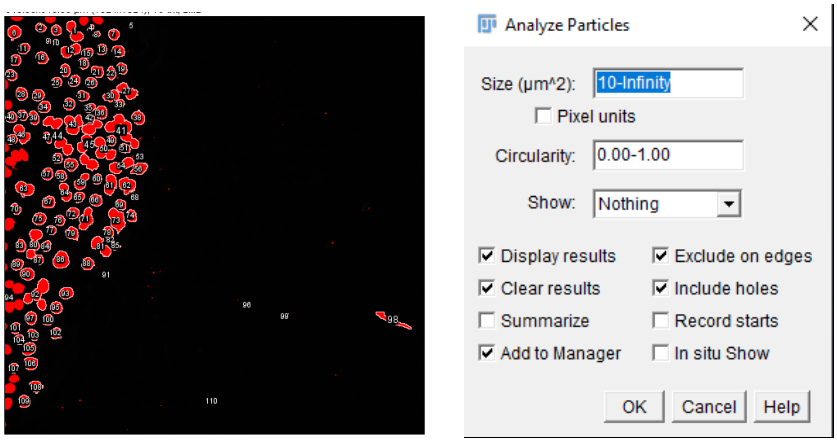


3. Create the mask for the image

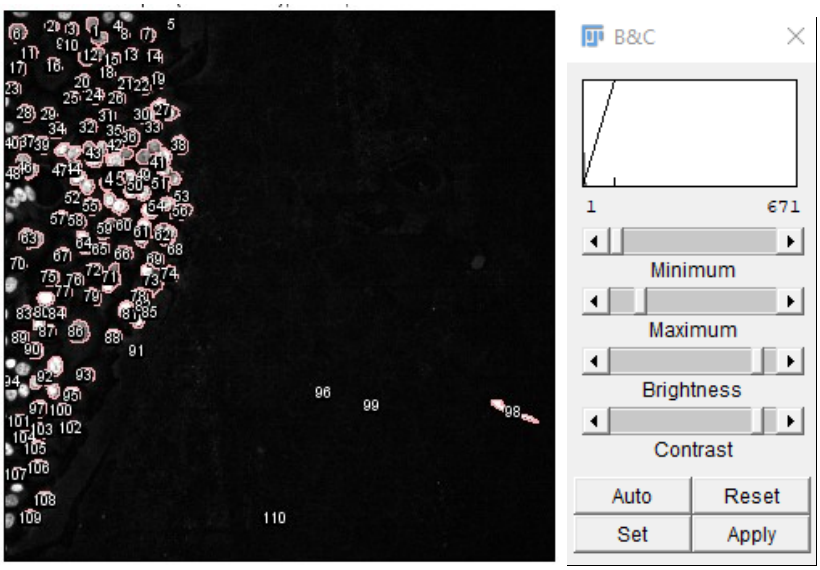
3.1 Open the first image from images - folder. Adjust the threshold (Image → Adjust → Threshold) using Huang method (or other suitable). Make sure you catch all cell nuclei.



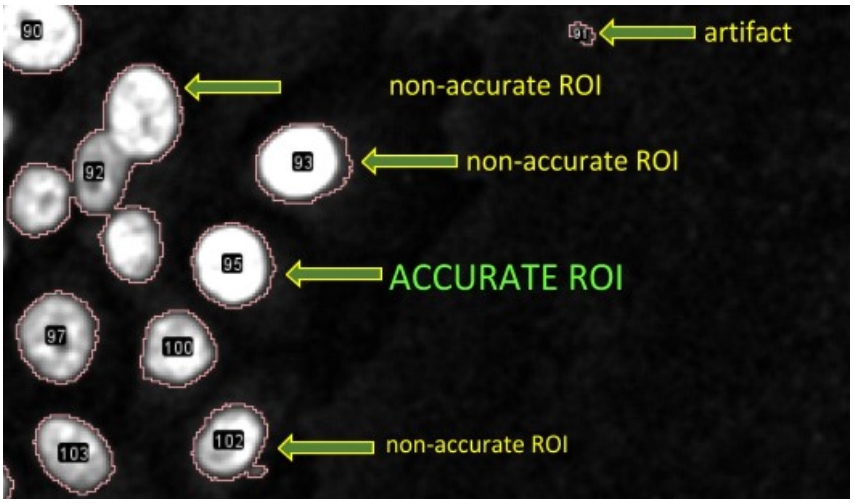
3.3 Create ROIs (Analyze → Analyze Particles)

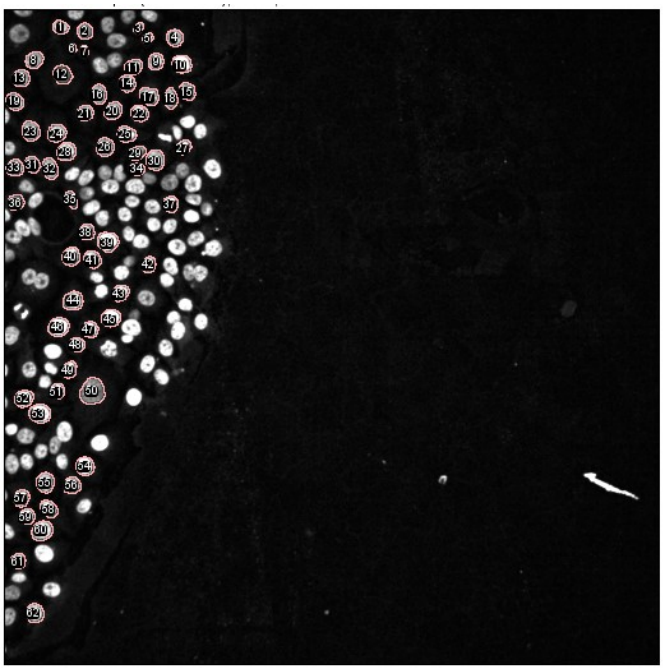


3.4 Adjust the brightness (Image → Adjust → Contrast/Brightness) lowering the Maximum value to see all cells in the image.

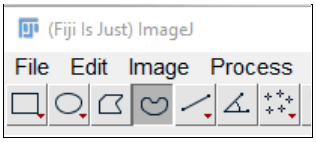


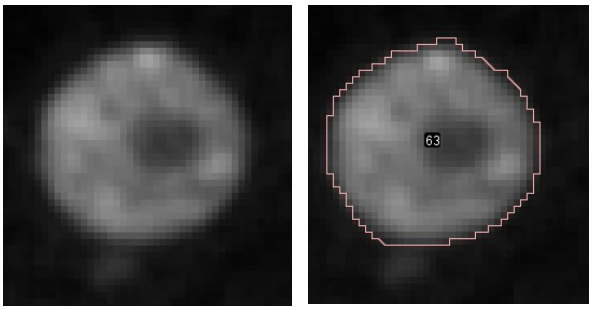
3.5 Remove all artifacts and non-accurate ROIs from the image. You may have to remove multiple ROIs to generate quality training dataset.



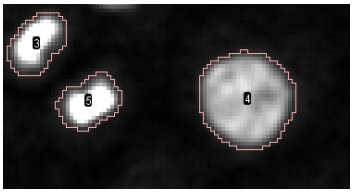


3.6 Draw ROIs for unlabeled nuclei with a drawing tool. Finally, press the t-letter to identify it. Draw ROI to all nuclei.

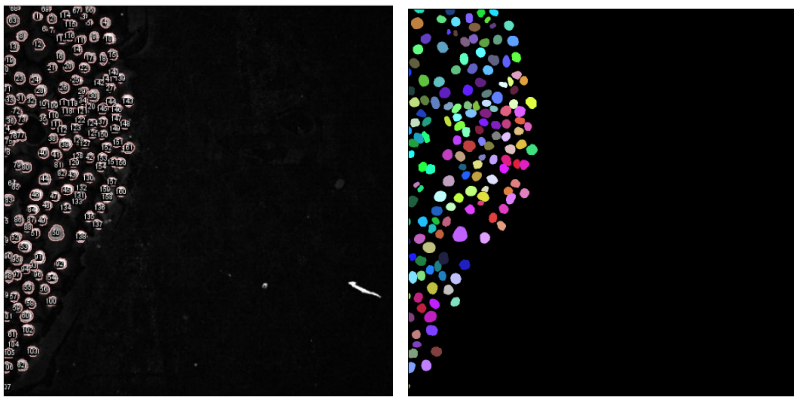




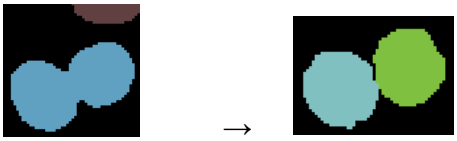
Also, include the just divided cells.



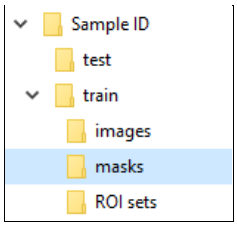
3.7 Fill ROIs to create mask (Plugins → LOCI → ROI map).

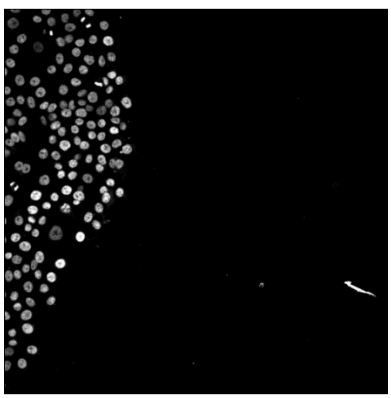


3.8 Check the mask. If two nuclei are merged remove the ROI in question and draw a new one. Fill ROIs again.



3.9 Save the mask with the same name as the corresponding image in tiff-format.





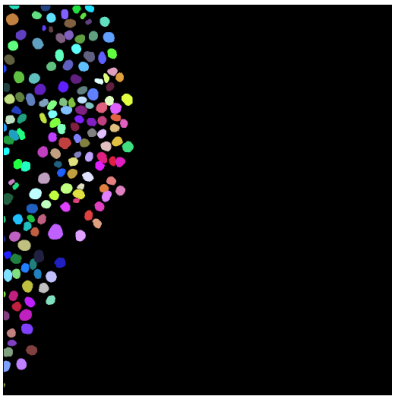
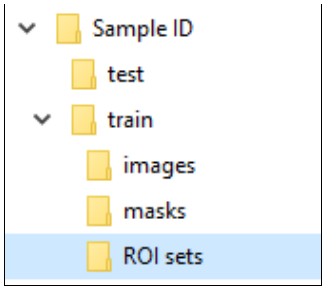


image 1.tiff (in images- folder) image 1.tiff (in masks- folder)

3.10. Save the ROI set (identify image in naming)



3.11 Create the mask for every image by repeating 3.1-3.10