Creating a platform for the democratisation of Deep Learning in microscopy

A dissertation submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

of

University College London

April 2022

Lucas von Chamier

MRC Laboratory for Molecular Cell Biology

Declaration

I, Lucas von Chamier, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

[redacted]

Lucas von Chamier, April 2022

Abstract

One of the major technological success stories of the last decade has been the advent of deep learning (DL), which has touched almost every aspect of modern life after a breakthrough performance in an image detection challenge in 2012. The bioimaging community quickly recognised the prospect of the automated ability to make sense of image data with near-human performance as potentially ground-breaking. In the decade since, hundreds of publications have used this technology to tackle many problems related to image analysis, such as labelling or counting cells, identifying cells or organelles of interest in large image datasets, or removing noise or improving the resolution of images. However, the adoption of DL tools in large parts of the bioimaging community has been slow, and many tools have remained in the hands of developers. In this project, I have identified key barriers which have prevented many bioimage analysts and microscopists from accessing existing DL technology in their field and have, in collaboration with colleagues, developed the ZeroCostDL4Mic platform, which aims to address these barriers. This project is inspired by the observation that the most significant impact technology can have in science is when it becomes ubiquitous, that is, when its use becomes essential to address the community's questions. This work represents one of the first attempts to make DL tools accessible in a transparent, code-free, and affordable manner for bioimage analysis to unlock the full potential of DL via its democratisation for the bioimaging community.

Impact Statement

One of the key tasks in this project was assessing the practical obstacles of existing tools in DL for the bioimaging community and developing a tool to overcome these obstacles. The reason why it was necessary to overcome these obstacles lies in the massive potential DL can have for bioimaging and microscopy. DL tools have already been shown to outcompete many existing bioimaging algorithms regarding their speed and accuracy. Hence, widening the user-base for these tools has the immediate impact that more users can benefit from this technology. The impact of this project is thus based on the simple assumption that a larger community with access to certain technologies can create a more significant research impact than a smaller community with such access. Here, the intended growth of the user base of DL is achieved by removing three key barriers to accessibility. First, the developed method, ZeroCostDL4Mic, is free and widely accessible via a web browser. Hence, it removes the financial constraint, which can still be a limiting factor for new users of DL as this usually requires procuring powerful and expensive computational equipment. Second, given that DL is a new technology in bioimaging and other fields using large quantities of data, and literacy in the community is relatively low, the impact of this research is specifically giving users confidence in navigating DL techniques. This is achieved by removing the requirement to read code and instead creating a completely code-free interface for DL. Thirdly, this work provides guidelines towards good practice, both in developing DL-methods and in their safe use in research. This will aid researchers following these guidelines to exploit and test DL tools in the most gainful way for their data. Specifically, this work emphasises the importance of quality control, highlighting essential aspects which have previously rarely been explored, which will be important in interpreting data created in DL algorithms and help create more robust DL models.

Since the work is accessible in a web browser and some of the implemented tools, are agnostic to data type, the ZeroCostDL4Mic platform can be used widely and is theoretically accessible, even beyond bioimaging, to anyone with a paired dataset. This means that the project is uniquely placed in a position where it is as accessible by academia as by anyone with a web browser. Given that the tool is aimed at audiences without an understanding of coding, this could help new users and laypersons alike gain insight into DL and associated research and may help spawn similar projects in other fields of research, education, or art.

Acknowledgements

The creation of the ZeroCostDL4Mic platform during this project was a collaborative effort and would not have been possible without the contributions of Romain Laine and Guillaume Jacquemet, each of whom made several notebooks and datasets used in this project and helped in many discussions to form the essence of the ZeroCostDL4Mic repository. I would also like to greatly thank my supervisor Ricardo Henriques who gave the inspiration for this project, for continued guidance and advice during the development of the project, its publication and during the writing of this work. All three were great role models for me during my PhD and while developing the ZeroCostDL4Mic project and have pushed me to become a better scientist in the process. Specifically, in the very long and frustrating review process of our paper during the middle of the pandemic, I was grateful for their optimism and belief in this project which in turn helped me to continue believing in its merit.

I want to thank Christoph Spahn for his help testing many of the notebooks in this project and his correspondence, specifically for highlighting problems and solutions for the fnet 3D method and its implementation, and for the provision of the datasets used in chapter 2. Furthermore, I would like to thank the Ivaska laboratory for providing me with their laboratory’s Google Drive account with unlimited memory which greatly facilitated the testing of the notebooks and saving results. I also want to acknowledge the many beta-testers of the notebooks used in this project, specifically, Eleni Karinou, Seamus Holden, Pieta Matila, Martina Lerche, Daniel Krentzel (who also created the 3D U-net notebook for ZeroCostDL4Mic repository), Sara Hernández-Pérez and also Johanna Jukkala who jointly began working on this project with me.

I would like to thank my thesis committee, Jonathan Chubb, Alan Lowe and Yanlan Mao whose comments during our meetings gave me perspective on the merit of my project and good advice in structuring my thesis.

I would like to express my deep gratitude to Julie Pitcher for her great personal support during a family emergency which threatened the completion of this work and which allowed me to extend the funding for this project.

I would like to thank Poppy Stockbridge who sacrificed a significant amount of time and sleep to care for our son so that I could work on this thesis. And Leander who

was born right in the middle of the writing process. Even though you cost me many hours of sleep and work, you brought a happiness into my life which helped me stay positive during the long hours of working on this thesis.

Finally, despite the lack of permission, I want to acknowledge the contribution of Henrietta Lachs for the cells descended from her body which were used in this work.

Table of Contents

List of figures

I. Introduction

I. 1. Literature Review

The latest technological advances have pushed the capabilities of microscopy in bioimaging so far that the remaining domains that elude imaging are small and increasingly difficult to access with current technologies. Hence, the bioimaging community is increasingly searching for the next novel technologies which can help overcome the hurdles for further exploration of cell biology via images. The recent advent of deep learning (DL) for image-based tasks from near-human performance in object recognition tasks[1,2] to applications in self-driving cars[3–5], has therefore quickly garnered interest in the bioimaging community as it promises solutions to significant challenges inherent in existing imaging and analysis methods. In this project, I aimed to create a tool to boost access to this novel technology for the bioimaging community. To understand the exact motivation for this endeavour, in this chapter, I will first give an overview of the two critical challenges in microscopy that limit the biological information that can be accessed in bioimaging studies. Next, I will outline why DL may be suited to solve these challenges, giving an overview first of how DL works, and second, which microscopy applications have been implemented by the community. In the final section, I will discuss why the democratisation of technologies in microscopy is crucial for their impact on research. This directly informs the primary objectives this project attempts to achieve, which are outlined at the end of the discussion.

I. 1. 1. Challenges of bioimaging

I. 1. 1. a. The resolution and toxicity dilemma

Since its invention, innovation in microscopy has been aimed at lowering the threshold of the smallest structures that could be visualised. This promises insight into previously unseen domains of life. Ultimately, microscopic imaging of the smallest components of living systems answers one of the most profound questions in biology: how the organisation of non-living objects on the atomic or molecular scale leads to the complexities emergent in living systems. The most recent innovations in microscopy have now come increasingly close to reaching this goal. Electron microscopy (EM), aided by innovation in sample preparation[6,7] and image reconstruction[8], today gives access, in fixed specimens, to biological structures down to the atomic scale[9,10]. To visualise subcellular structures in living cells, fluorescence microscopy boosted by a revolution in the form of fluorescent proteins[11], can give insights to detailed subcellular features which would be invisible under white light. At the turn of the century, different fluorescent microscopy techniques, most notably, stimulated emission depletion (STED)[12], structured illumination microscopy (SIM)[13] and single-molecule localisation microscopy (SMLM)[14,15], were developed which achieved unprecedented resolution for optical microscopy methods. In this context, resolution can be defined as the smallest distance at which two real objects can still be separated in an image. In optical microscopy resolution is limited by the self-interference of light waves in the optics of microscopes. Point sources of light thus get converted to a point-spread function which appears as a blur or multiple blurry rings around the centre of an object on the imaging plane, a phenomenon known as an Airy disk. The shape and extent of the PSF and Airy disk depends on the optics of the microscope and the wavelength of the light used for imaging. However, because of this phenomenon there exists a limit for a given wavelength and imaging system where two neighbouring object's PSFs will overlap and any distance between the objects will not be resolvable on the imaging plane. This generally prevents structures from being resolved which are smaller than about half the wavelength of the light used for imaging[16]. Using light in the optical spectrum meant that structures below roughly 250nm could not be imaged with light microscopes, precluding most protein complexes and many subcellular structures from visualisation. With the above fluorescence microscopy techniques, collectively termed super-resolution microscopy (SRM), structures below this limit, with dimensions in the molecular scale, could be resolved for the first time using optical methods, for example, in the organisation of cytoskeletal structures in neurons[17] (Fig. 1).

When considering only resolution, it could be assumed that microscopy techniques today have nearly reached the ultimate goal of bioimaging, that of visualising life down to its fundamental particles. However, it has become increasingly clear that visualisation of life to this level requires another key aspect aside from spatial resolution, which is capturing its dynamics over time. Current EM and SRM techniques both fall short in achieving this satisfactorily. EM is inherently toxic as it requires samples to be fixed and therefore dead, to be imaged[18]. SRM, as most fluorescence microscopy techniques, suffers from the phenomenon of phototoxicity[19]. Phototoxicity describes the detrimental effect of light on cell homeostasis, which can alter natural cellular behaviours and even lead to apoptosis of the imaged cell[20] and can arise from several sources, often dependent on the type of light used in imaging. Notably, in cells labelled with and imaged for fluorescent proteins or dyes, the activation of the fluorophore can produce radical oxygen species (ROS), which trigger apoptotic pathways or disrupt cell homeostasis[19,21]. Imaging with short wavelengths such as UV light risks light-induced DNA damage if UV light is absorbed in the nucleus[19,20]. This can initiate DNA repair pathways and apoptosis. Although longer wavelengths such as near-infrared light have been observed to be less damaging to cells via direct phototoxicity[19], such light sources can yet lead to sample heating and could potentially lead to heat shock pathways that disrupt cell homeostasis[22].

Phototoxicity scales with the dose of light transmitted onto the sample, i.e. the number of photons absorbed per square area per unit time[22]. Hence, phototoxicity can be reduced by reducing the intensity of the incident light (number of photons), reducing the exposure time per acquisition or, in time-lapses, reducing the number of frames per time unit[19,20,22,23]. Furthermore, since ROS production is primarily related to the excitation of fluorescence, reducing fluorescence, e.g. by imaging in brightfield microscopes, should reduce the impact of imaging on the specimen. However, reducing the number of excitatory photons or not inducing fluorescence also reduces the number of photons emitted from the sample to the signal detector of the microscope's camera[24]. This leads to a decrease in an image's signal-to-noise ratio (SNR)[2], as the intensity of background noise from out-of-focus objects, the fluorescence of the growth medium or shot noise on the camera's detector become significant in comparison with the signal collected from the sample. This is problematic as high SNR is required to visualise delicate structures in cells and is essential for SRM techniques to achieve sub-diffraction limit resolution[25]. The impact of phototoxicity is most significant in recording time-lapses as each acquisition increases the light-dose experienced by the sample[22,25]. Reducing phototoxicity in time-lapses without sacrificing SNR per image requires spacing acquisitions to a period in which the sample can recover, which effectively reduces the temporal resolution of the timelapse[20,22,23] (Fig. 1A). An additional problem is that phototoxicity induced pathways may occur silently in cells that appear healthy and toxic effects may only be discovered after the cell undergoes morphological changes[23]. Therefore, the true impact of phototoxicity, especially in fluorescence microscopy, can be underestimated in many imaging experiments and even raises the question of whether the behaviours shown by cells imaged for sustained periods under fluorescence can be deemed ‘natural’ and thus of biological interest.

Phototoxicity is a crucial caveat of most SRM techniques. Techniques such as STED and SMLM rely specifically on very high-intensity lasers to achieve sub-diffraction limit resolution[12,14,15], making them significantly more toxic to cells than diffraction-limited microscopy methods[19,25]. The other major SRM technique, structured illumination microscopy (SIM) does not usually require as high-intensity illumination of the sample as other SRM methods. However, each SIM-resolved image requires multiple acquisitions with patterned image filters which are used to computationally reconstruct the super-resolved spatial features on the image[13]. This increases the total light dose required for a single super-resolved image when compared to standard microscopy where one acquisition is needed per image. The increased phototoxicity related to SRM makes it extremely challenging to exploit the potential of super-resolution for most biological processes beyond the timescale of minutes[25] (Fig. 1 B). Although acquisitions with SRM, such as STED, over hours, have been reported with apparently healthy samples[26], potential phototoxic effects are not usually quantified in these studies, making it difficult to determine if ‘silent’ phototoxic effects are affecting the sample in unknown ways[19,23].

The inability of the best optical microscopy methods to exploit their resolution for longer acquisitions of living structures leaves a white space in the domain of cellular processes which can be imaged (shown in red in Fig. 1B). Closing these gaps for possible exploration is one of the key drivers for developing new methods. This has led to the development of increasingly complex techniques such as lattice-light sheet microscopy (LLSM)[27,] which is specifically designed to optimise the light-dose for 3D acquisitions over time. Another approach is to optimise existing SRM techniques by using more sensitive detectors[28,29] or pushing the optical parameters of lenses[30,31]. However, most technological innovations which can push the envelope of resolution and live-cell compatibility are technologically complex or expensive while providing only limited improvements to existing technology. Therefore, computational approaches that tend to be inexpensive are increasingly sought after in the bioimaging community to help solve challenging imaging problems[32].

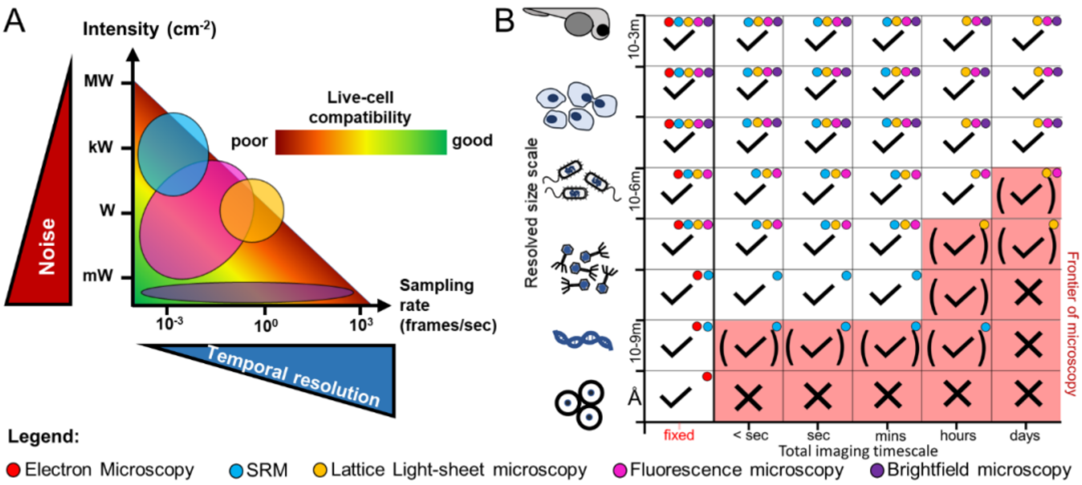


Figure 1 - Boundaries of microscopy in live-cell imaging – A) *Schematic of the dilemma between image quality (noise), possible temporal resolution and live-cell compatibility.* ***B)*** *Domains of biology that can and cannot be visualised in time and space with current microscopy techniques. Phototoxicity limits high spatial and temporal resolution at longer timescales (see A). The red area indicates the current limitations of microscopy and, thus, the domains that DL methods could help access. The coloured circles indicate the microscopy methods used for each size and timescale. Symbols: tick – size scale at timescale is acquirable with current techniques, tick in brackets – size scale at timescale can under some circumstances be visualised in specific samples with cutting-edge equipment, x – size scale at timescale cannot currently be visualised.*

I. 1. 1. b. Advent of high-throughput imaging and analysis bottlenecks

The use of software methods in microscopy has expanded in line with the increase of storage space and availability of cheap, powerful, and fast computers during the last three decades. These technological advances have enabled new forms of imaging studies to be conducted. High-throughput microscopy studies containing tens of thousands of individual fields-of-view (FOV)[33,] which allow large scale drug screens[34–36] with hundreds of conditions, only became possible thanks to increased capabilities of computational equipment. Similarly, faster and more efficient storage made it feasible to collect images with extremely large dimensions to resolve small structures present in extensive fields of view[37] or 3D stacks[38] or to image[38,39] and annotate[40] entire organs.

Automation of image collection and more efficient data storage leading to significant volumes of data made it necessary to create more sophisticated analysis methods to extract information from these datasets. This led to rapid growth in bioimage analysis, which is concerned primarily with the analysis of biological image data, rather than its acquisition and is beginning to be treated as its own discipline in biology[32]. In analysis, computational methods have the advantage of improving the precision and scale at which data can be analysed[32]. This has contributed significantly to modern imaging studies and the data types that can be gainfully used in research. Techniques like RNA FISH[41,42], cell-tracking[43–45] and lineage tracing[46,47] and SRM techniques like SMLM[14,15] or SIM[13], would not be feasible without using computational image analysis post-acquisition. Bioimage analysis has also addressed the previously discussed microscopy problem of phototoxicity and resolution. Methods such as super-resolution radial fluctuations (SRRF)[48] or image deconvolution[49,50], which models and then removes the noise created during imaging, offer alternatives to the optical super-resolution methods and do not require the light dose often required for SRM. Hence, bioimage analysis can have an impact on microscopic imaging challenges that is comparable to innovation in optical methods. However, in contrast to innovations in microscopy, such as SRM techniques or LLSM, bioimage analysis innovation is easily disseminated and comparably inexpensive, as it is based primarily on code that can be shared on the Web, reaching more users than many microscopy techniques[51]. Since many researchers who require image analysis tools possess limited experience in coding environments, developers have a great incentive to facilitate the use of software tools by making them accessible, for example, through graphical user interfaces (GUI) that require no code interaction[51–54]. An example of such a tool is the ImageJ/Fiji platform, which researchers widely use to assess many different types of data and has a highly active developer and user base[53] which also contributes to the fast improvement of methods and resolution of problems.

However, the advent of cheap, fast, and efficient data acquisition has come with new challenges. One significant problem lies in replicating the abilities of humans to identify substantial features in images at the scale of the data volumes produced today. For example, van Valen et al. note that the segmentation of nuclei, which is necessary, for example, to track nuclei in time-lapses or to measure their volume, can take hours to days of work[55]. This type of work is trivial but strenuous for a human who could likely perform more impactful tasks if such activities as nuclear segmentation could be automated. However, humans have evolved a pattern recognition capacity that most algorithms still cannot replicate. Hence, trivial tasks such as separating the background from foreground objects in images automatically remains a significant challenge in computer vision[56]. More involved tasks such as identifying different classes of objects are even more challenging to perform with standard bioimage analysis tools but are of significant value to biologists as complex phenotypes or patterns could be quickly identified by a machine rather than a human, which is beneficial especially in datasets with thousands of images[1]. Due to these difficulties, the ability to acquire and store very large datasets and images has not scaled at the same rate as the ability to analyse these datasets with human precision[57]. Extracting all informative biological features present in imaging data remains one of the key challenges of bioimaging[56,57] (Fig. 2).

Further hindering the potential of automated analysis of images via algorithms is that humans invariably introduce their own bias into code. For example, most algorithms designed to remove noise or enhance resolution use error estimators (also named priors), which are essentially based on an educated guess on the nature of the image noise[58,59]. This can lead to artefacts, such as edge smoothing[59] or the introduction of artificial structures on the image, which is relatively common in some SRM techniques, such as SMLM[60,61] or SIM[62,63]. Hence, images obtained by algorithms based on their creators' assumptions are less agnostic than the raw images produced by microscopy, which can make them less valuable in research than the latter.

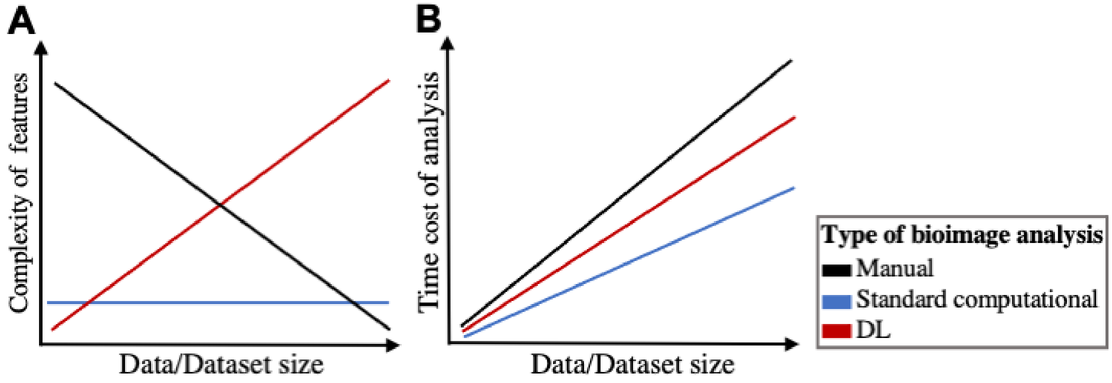


Figure 2 - The promise of DL in bioimaging – comparing the potential of DL qualitatively with manual and standard computational bioimage analysis: A) *As datasets become larger and more complex, it becomes difficult or impossible to annotate by hand. However, human annotation is usually unsurpassed for individual images, i.e. small dataset sizes. Current bioimage analysis methods can be algorithmically programmed with relative ease and can extract key features from datasets, but often not at the level of a human analyst. Creating a DL pipeline requires the same or more effort than standard algorithms, as they require dataset curation and training, and they tend to not perform well when little data is available. However, as the size of the available training data increases, so usually does the performance of DL tools. Even complex tasks can be performed on large datasets with well-trained DL-methods which would be difficult or impossible to do manually or with standard methods. B) DL tools may be slower than standard methods, specifically when the computational resources are not available for training or prediction. However, for tasks such as manual labelling or classification they are significantly faster than a human.*

I. 1. 2. Deep Learning (DL) as a solution to challenges in bioimaging

I. 1. 2. a. Overview of DL technology

The difficulty of replicating human performance in bioimage analysis tasks, such as object recognition, and the introduction of human bias, is not a problem unique to microscopic images but has been a long-standing puzzle in computer science. The first algorithm reaching human-like performance in classifying images on a large and diverse image dataset was published in 2012[2]. In this breakthrough, a team from the University of Toronto used an artificial neural network (ANN) called AlexNet to participate in a popular object identification competition and easily outcompeted the nearest competitors in terms of precision and even matched human annotators in this task[2].

This result sparked a massive interest in ANNs, which now dominate other algorithms in many tasks from image annotation[64,65] to language recognition[66–68]. The sudden success of ANNs in such data challenges was surprising since ANNs are a relatively ‘old’ technology, known since the beginning of computer science[69], however, with limited use, since their architecture makes them computationally expensive to use[66]. To understand why ANNs can be so resource-intensive, it is helpful to understand how they operate, which I will outline below.

i. Artificial neural network (ANN) architecture

As their names suggest, ANNs were originally indeed an attempt to represent neural networks found in the nervous systems of complex organisms. Since these networks perform complex functions despite consisting of relatively simple units, the neurons, it was thought that the functions of biological neural networks could be replicated artificially by mimicking neuronal activity with a simple model[70]. Such a model of a neuron, termed the perceptron, was developed in 1957 by Rosenblatt[69]. The perceptron takes an arbitrary number of inputs, performs a function operation on these inputs and gives an output. In this process, different inputs can be weighted differently, similarly to how some synapses can have stronger influence on a neuron’s activity than others. Depending on a threshold value, this output is converted by the perceptron to either a 0 or 1, making the perceptron essentially a binary classifier of a weighted sum of inputs. Due to this simplistic nature, an individual perceptron cannot perform operations beyond simple linear classifications. To achieve more complex non-linear classifications, perceptrons can be stacked in ‘layers’ with each layer consisting of multiple perceptrons, and each perceptron’s output becoming the input to a downstream perceptron (see Fig. 3A). Modern ANNs are essentially an adaptation of such ‘multi-layer perceptrons’. And even though a perceptron, can in no way model the true or even relevant complexities of real neurons (indeed, these complexities are not even clear biologically), in such networks, they are sometimes referred to as ‘neurons’. Even though this might be a misleading nomenclature, since a perceptron is very much not like a neuron, it is arguably a more intuitive and descriptive term than perceptron, and in the following chapters, I will continue to use it instead of ‘perceptron’ despite the obvious caveats.

As in the multilayer-perceptron, an ANN has at least an input and an output layer, but further ‘hidden’ layers can be added between to give rise to ‘deeper’ networks. This has led to the term ‘deep learning’ (DL) when referring to learning methods that use ANNs with several hidden layers (Fig. 3A). Each neuron receives inputs from one or more neurons of a preceding layer and transforms this input using a mathematical function called an *activation function*. The output from this computation is passed to neurons in the next layer after multiplying the value with a weight. The weights of an ANN are flexible and are the main parameters that allow the network to adjust to a given dataset, as they can reinforce the output of specific neurons and suppress others[71].

The most widely used type of ANN for image-related tasks, and also the type of network used by AlexNet[2], are convolutional neural networks (CNN)[66,72]. In CNNs, the so-called convolution layers contain sets of convolution matrices (kernels) which can extract specific types of features or patterns from the input image, resulting in different representations of the image known as feature representations or feature maps (Fig. 3B). Each convolutional layer will output multiple feature maps, the number of which is equivalent to the number of kernels in the layer. In most CNNs, convolution layers are interspaced with pooling layers which reduce the dimensionality of the feature maps. This is usually achieved by assembling a smaller image from individual pixels of the feature map, by sliding a window of a specific size (e.g. 3x3 pixels) over the feature map and selecting only one pixel from each position. A common strategy is choosing the pixel with the highest intensity from each window position, called *max-pooling.* Pooling is used to simplify features while reducing the size of the feature maps, which reduces the memory footprint of each image.

After multiple layers have processed an input image, the feature maps are flattened, which means that the pixels in the feature maps are arranged as a one-dimensional vector[71]. This vector is passed through so-called *fully connected* layers. In fully connected layers, each neuron is connected to every neuron in the following layer. For classification tasks, the final fully connected layer will use an activation function resulting in the sum of all values in the output vector to add up to 1 (Fig. 3B).

ii. Training and applying ANNs

The network has not been trained for any task up to this point. To train the network, say for object classification, the output from the CNN needs to be compared to a reference, also called a *ground-truth* or *target*, against which it can assess its performance[73]. The goal of training the CNN is to adjust the weights applied to each kernel in the convolutional layers of the network, such that the CNN can find a representation of an input image that is as close as possible to a ground-truth. For classification tasks, the target will usually a be vector where each entry represents a class[66]. These entries will commonly be binary and define a class as present (1) or not present (0) in an image (Fig. 3C). The difference between the output of the CNN and the ground-truth is assessed using a *loss function.* This loss function is a function of the weights of the CNN and evaluates the mismatch between input and ground-truth. To improve its performance in representing the ground-truth, the weights of the CNN must be updated to reduce the value of the loss function. To achieve this, the contribution of each weight to the final value of the loss function needs to be evaluated, which is done using the so-called error *backpropagation*[74] (Fig. 3C). In backpropagation, the derivative of the loss function is first evaluated with respect to the weights in the final layer of the CNN and proceeds backwards through the network layer by layer. Since the error contributed by each weight depends on the weights of neurons in previous layers, the derivative calculated for previous layers is carried out according to the chain rule of calculus[74].

Once the partial derivatives of the loss function for each weight of the network are known, the weights can be updated to move down the slope of the derivative, using a process called *gradient descent*[75] (Fig. 3C). There are different strategies for gradient descent, but all methods have the goal to adjust the value of a weight so that the derivative of the loss function will have a lower value than before the adjustment. The magnitude of the adjustment change is determined by a parameter termed *learning rate*. This value determines how significantly the gradient descent will change the weights in the CNN. Large learning rates rapidly shift the value of the gradient and will change the loss of the network significantly[76]. However, large learning rates may change the value of the loss function so much that it cannot converge to a minimum that represents the optimal state of the weights for a given dataset. A very small learning rate may allow the loss function to converge slowly or at a local rather than a global minimum. Ideally, the learning rate takes a value that balances the need for convergence and the ability to skip beyond the local minima of the loss-function space[76,77].

During training, the CNN will be presented with many input-target pairs, each of which will update the weights via calculation of a loss function and subsequent backpropagation and gradient descent. In many training pipelines, this process is repeated several times on a given dataset since CNNs' convergence towards low losses tends to be gradual and thus slow. For convenience, the training process is often split into training *'epochs'* during which the CNN is presented with each training-input pair once (or a specified number of times). Calculating the loss of the network after each epoch is a convenient way to follow the progress of the network throughout the training process[78].

The loss decreases as the network improves its performance until it ideally converges towards a minimum. Once the user deems the network sufficiently trained (either after the loss stops improving sufficiently or at a specific threshold value of the loss), it can be used on an 'unseen' dataset, i.e., a dataset without ground-truth targets. The trained network can now apply the learned transformation from inputs to targets on unseen inputs, and '*predict*' or '*infer*' an *output* or *result*[73]. For this use case, the weights remain fixed at the value that produced the lowest loss during training, thus no longer changing via gradient descent. From here on, such trained networks will be referred to as *models*. An important feature of DL models is the ability to '*generalise*' to novel data which describes how well-trained models can apply their learned mapping to a dataset that was not part of the training set. When the model can perform the desired image processing well on such 'unseen' data, and specifically when this data is very different to the training dataset, it is said to generalise well. Conversely, neural networks can also underfit to training datasets. This can occur when the model's own complexity, e.g. in terms of the number of learnable weights, is smaller than the number of features present in the training dataset. In this case, training losses are expected to not converge at all, and the model learning no relevant features in the dataset. In the era of very broad and deep neural networks this phenomenon however is much rarer and less well explored than underfitting, and can be remedied for example by adding more neurons to the network's hidden layers[79].

However, it is common for DL models to fail to generalise appropriately, resulting in unrealistic or inaccurate predictions[80]. This is often caused by a phenomenon called *overfitting,* which occurs when an ANN learns a specific input-target transformation so well on a training dataset that it cannot generalise the transformation to data beyond this dataset. Preventing overfitting is a common task for DL practitioners and can be remedied in different ways. The first is to identify overfitting during training and before using the network for predictions. This can be done by setting aside a *validation dataset* which can be a random sample from the training dataset[78]. The validation dataset does not lead to an update of the CNN's weights but is instead used after each epoch to evaluate how well the CNN can perform on an image that is not used or 'seen' during training. The loss calculated between the network's prediction from the validation input and the validation target is known as the *validation loss*. Calculating the validation loss simultaneously to the training loss at the end of an epoch can help identify overfitting during training in the following way: while validation loss and training loss decline at roughly the same rate, the model likely does not overfit. When the training loss continues to decrease with each epoch while the validation loss stays the same or rises, this indicates overfitting, since the model's performance is no longer generalisable to the 'unseen' validation dataset.

Once a network is trained, it can be applied for tasks on 'unseen' data, i.e. data that the user is interested in analysing and for which presumably no ground-truth exists. However, there are some non-trivial challenges in this step which are connected to the limited interpretability of DL-model predictions. Like other computational image analysis methods, CNNs can predict artifacts or detect incorrect objects (false positives) or miss existing objects (false negatives). Without availability of a ground-truth against which to compare a model's predictions, such mistakes can be difficult or even impossible to identify for well-trained models. These problems can even be compounded by some methods giving high ‘confidence’ scores for predictions which may be incorrect. These confidence scores do not reflect the actual probability of predictions’ correctness, but rather are an internal measure of models ranking different object categories. The reasons for such failures are still subject of research, partially because there is no full theory of how the complexity of neural networks contributes to its performance (though this understanding is improving[81,82]). However, there are methods which can improve the interpretability of model outputs which can also compensate for these problems in neural network predictions. For instance, different methods exist which predict probability distributions rather than single categories where models’ confidence in a prediction is reflected in the variance of the distributions around individual predictions. Unlike the confidence score given by models, these measures reflect a more accurate description of the probability of a correct prediction. This can help users identify when model outputs become unreliable, even in the absence of a ground-truth label. However, the methods which allow such probabilities to be given by a neural network are often computationally expensive, in training (see e.g. Bayesian neural networks[83,84] and ensemble learning[85]) or during inference (e.g. test-time augmentation[86,87]). They are also not implemented in many of the most popular methods, and thus not accessible to general users of the most powerful DL-methods. For general use-cases, the gold-standard to determine the quality of a DL-model thus remains a test against a ground-truth dataset in a quality control step. This quality control (or test-) dataset contains input-output pairs that are not part of the training dataset but can be picked from the same population of images, i.e. this dataset can be set aside prior to training from the training dataset. To test the model's performance, the predictions of the model on the test inputs are compared to the test targets. The comparison can be made using similarity or quality metrics which measure the errors between prediction and ground-truth. Which metric is used depends on the type of task performed by the network. Quality metrics can give a quantitative assessment of DL-models and are the main way in which different models are ranked against each other in competition[88,89] and in the literature.

iii. Potential challenges and solutions to dataset curation for DL

Due to its reliance on training datasets, DL performance fundamentally depends on the features present in the datasets used for ANN training. Therefore, neural network performance scales strongly with the size of the dataset and networks trained on larger datasets perform better than those trained in smaller ones[2,90], as they adjust to the diversity of features present in large datasets. For the same reason, it is important to avoid patterns in the dataset that may introduce bias. While in standard algorithms bias may be introduced by assumptions or priors introduced by developers, in DL bias can occur due to imbalanced features in the dataset. For example, Moeckl et al. showed how the shapes present in a training dataset in a simple task of estimating the shape of objects on an image determined the model's predictions significantly[91]. If the training dataset contained only rectangles, the trained model predicted only rectangles in unseen images, even if the ground-truth consisted of other shapes. Conversely, a diverse dataset with multiple shapes allowed it to identify shapes in an unseen dataset correctly. This example shows how creating a suitable training dataset is crucial for creating useful DL models.

1. Data Augmentation

The reliance on large datasets is often regarded as a bottleneck in DL. Many networks with the best performance for object detection, which have made DL popular, were trained on very large public datasets[92–95]. Such datasets are rarely available for users who want to use DL on their data and annotating or curating such training datasets at a scale similar to those in image challenges can be difficult or impossible. Therefore, several tricks are commonly employed within DL methods to overcome the hurdle of dataset size. The first is to augment the dataset. This means that each image in a dataset is slightly changed by some form of image manipulation, such as rotation, and this transformed image is added to the dataset[96]. Using ‘geometric’ image transformations like rotation, shearing, shifting, mirroring, or zooming or manipulating the image’s signal such as noise, contrast, or colour changes can quickly increase the size of a dataset by nearly arbitrary factors[96,97]. Another popular approach is to extract patches or crops from images, which can, depending on the size and number of patches per image, increase the total number of images in the dataset[96]. Neural networks theoretically do not recognize transformed replicates of an image as the same image since the values of its pixels and the structures shown do not match the original. Thus, a dataset can be conveniently expanded using augmentation without requiring the user to produce more data[80]. However, there are some limitations; for example, if the augmentation changes the appearance of objects the user wants a model to recognize, specific transformations need to be avoided. Augmentation can also increase the contribution of unique features in the dataset, leading to worse performance on a test dataset than a model trained on a non-augmented dataset[97].

2. Transfer Learning

Another approach is to not train neural networks from scratch on new datasets. The early layers of neural networks may learn surprisingly similar representations from datasets, as these layers tend to identify simple patterns. Hence, using a well-trained network from a different dataset as a starting point can give users with smaller datasets the ability to fine-tune a well-rounded network quickly to their dataset. The process in which ANNs are used from a trained starting point to learn a new task is called *transfer learning*. Pre-trained models are commonly used as starting points in many published object detection tasks. For example, a ResNet[65] pre-trained on a large imaging dataset has been used in many object detection publications, which detect completely different types of objects[1,98–100]. Transfer learning can be powerful in many cases and save the user time and effort with their dataset curation and training time to reach the desired performance. It is also possible to train only some layers of a pre-trained ANN for this purpose. Especially when models were pre-trained on very large datasets, it can be sufficient to train only the final layer of the network, which is used to create the class labels of the dataset. This approach reduces the training time for the network and can still reach strong performance for new classification tasks[101,102].

A potential drawback of using pretrained models is the risk of introducing unintended bias from training datasets into the new task. For example, many networks pre-trained on public datasets use data from the web, which is known to be unrepresentative of real-world diversity[1,103,104]. This can lead to an intransparency of models' performance: if the model fails to adjust to a new task, is this due to problems in the original or the new dataset? Further, the model could predict images with artefacts based on the original dataset, which may be easy or, in a worse case, difficult to spot by the user.

3. Unsupervised Learning Strategies

Augmentation or the use of pre-trained models are popular approaches to overcome the sometimes laborious task of dataset curation, with ground-truth targets for each input image. This type of dataset is necessary for the training described above, called supervised training (also ‘fully’ or ‘strongly’ supervised). Most DL models used today are trained using this approach. Due to the limitations of curation, developers of DL methods have been interested in alternative approaches that range from weakly supervised training[105–107], where only a subset of training images has ground truth labels, to fully unsupervised training[84,108,109], where the network learns a task without any ground truth information. These significantly reduce the effort for dataset curation or remove the requirement entirely. Currently, existing approaches in the literature are relatively unique and are designed for particular data challenges, and thus challenging to generalise to broader tasks. Hence, unsupervised training methods are not as versatile as supervised methods, which may explain why they are not yet as widely used. Because of the appeal of not requiring the same effort in dataset curation, specifically for very large datasets, unsupervised training of DL models remains a highly anticipated innovation for many users of DL.

iv. Note on DL hardware

The impact of ANNs on the computer vision field has been dramatic, and they have become state-of-the-art in many image-processing tasks since the presentation of AlexNet. This impact was possible due to an important innovation by AlexNet developers, the use of *graphical processing units (GPU)* to perform the massive amounts of computations necessary for the backpropagation and gradient descent algorithms[66]. GPUs were originally developed to accelerate graphics processing for gaming computers and achieve this through massive parallelisation of calculations performed simultaneously. This parallelisation can improve the speed of calculations by several orders of magnitude and has been crucial in making the time required to train a neural network feasible for many users. Prior to GPU acceleration, ANNs could take days[110] for training on comparably trivial tasks and be deemed unfeasible for more complex image analysis tasks[66].

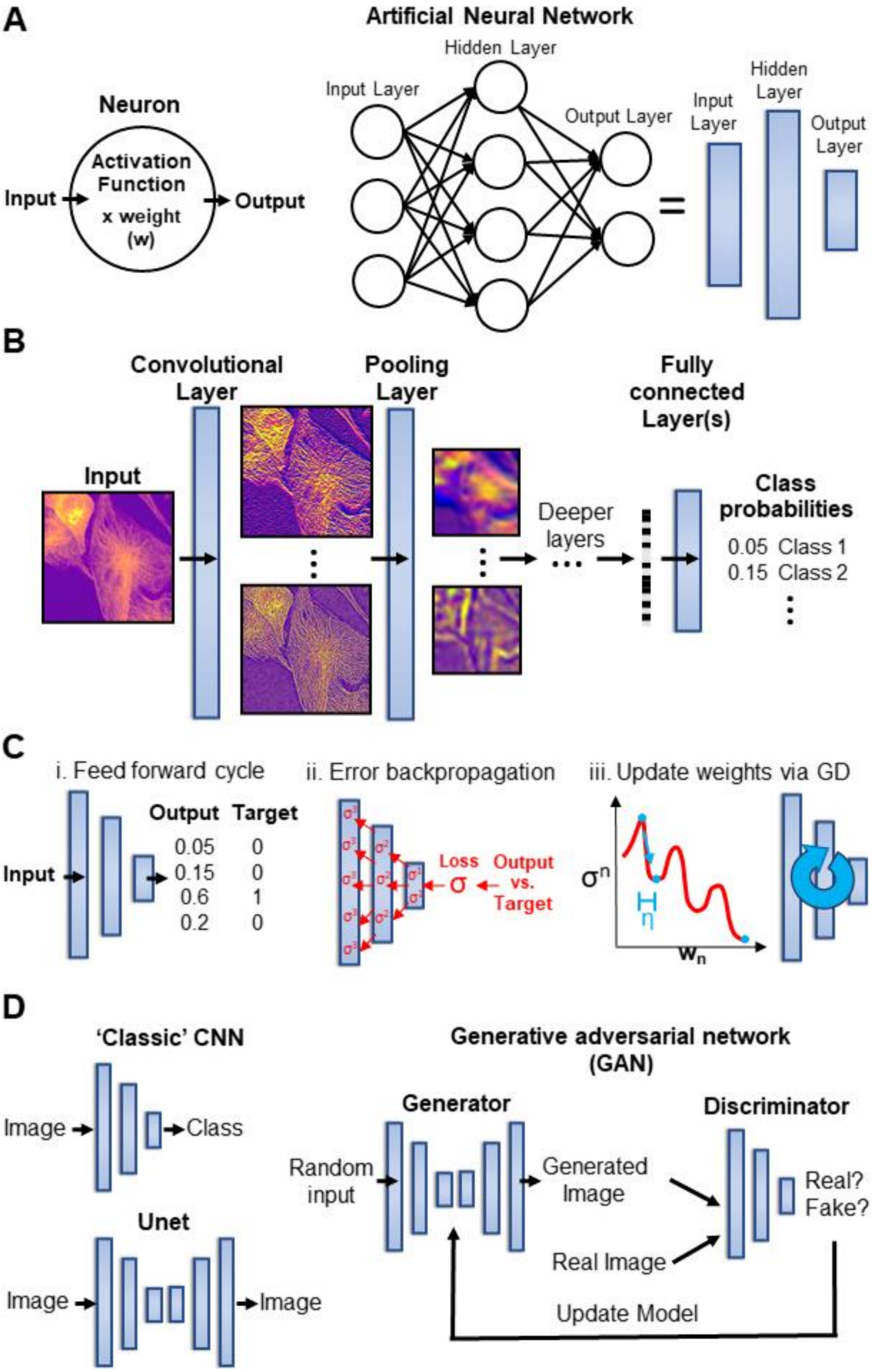

A
Neuron
Input
Activation Function
x weight (w)
Output
Artificial Neural Network
Input Layer
Hidden Layer
Output Layer
=
Input Layer
Hidden Layer
Output Layer
B
Input
Convolutional Layer
Pooling Layer
Deeper layers
Fully connected Layer(s)
Class probabilities
0.05 Class 1
0.15 Class 2
C
i. Feed forward cycle
Input
Output
Target
0.05
0
0.15
0
0.6
1
0.2
0
ii. Error backpropagation
Loss
σ
Output vs. Target
iii. Update weights via GD
σ^n
η
w_n
D
'Classic' CNN
Image
Class
Unet
Image
Image
Generative adversarial network (GAN)
Generator
Random input
Generated Image
Discriminator
Real Image
Real?
Fake?
Update Model

Figure 3 – Overview of DL methods and functions – A) *Neuron and basic artificial neural network (ANN) and a common schematic representation of an ANN.* ***B)*** *Basic function of a convolutional neural network (CNN) for classification, exemplified on a fluorescence microscopy image. CNNs have convolutional layers which create feature maps of an input image. Each feature map is simplified in pooling layers by combining pixels of specified image patches by using only certain pixels, e.g. the highest value pixel (max-pooling). In further convolution and pooling layers, the feature maps become increasingly abstract until they are flattened to a one-dimensional array, similar to a barcode passed through one or more fully connected layers that map the patterns in the flattened vector to classes by assigning a probability to each class.* ***C)*** *Three basic steps of ANN training: i) Feedforward cycle, as in B. At the end of this step, the output is compared with a provided target (ground truth) that represents the expected or desired outcome the network should provide. ii) The difference between the network output and target is used to calculate an error (loss) sigma. The contribution of each neuron in the ANN is calculated via backpropagation. iii) each neuron's weight (w_n) is updated via gradient descent so that its loss (sigma) follows a downward slope on the loss space (simplified to 2D). The step size of the gradient descent in w is given by the learning rate nu. Once all weights are updated, the steps are repeated for a new image or batch of images.* ***D)*** *Three commonly used types of ANN in bioimaging. 'Classic' CNN architecture is used for classification tasks, such as AlexNet. The U-Net architecture uses upsampling after the standard CNN layers which can be used for image-to-image tasks. A GAN network often contains both U-Net-type networks to generate images and classifiers (discriminators) to decide if a given image is created by the generator or is contained in the training dataset. The aim is to train the generator until the discriminator cannot distinguish 'fake' from 'real' images. The generator can then be used to generate artificial images that appear realistic within the domain of the training data set.*

I. 1. 2. b. DL in microscopy and image analysis

The effectiveness of ANNs in computer vision tasks quickly led to interest from the microscopy and bioimaging community, which was further fuelled by additional innovations in the ANN architecture. One of the most significant ones was the introduction of U-Net by Ronneberger et al.[111]. This architecture consists of a classic CNN without the fully connected layers which, instead of classifying images, uses an up-sampling pathway that reconstructs an image output instead of a class label. This allows U-net-type neural networks to be easily trained with image pairs as training examples and was shown by Ronneberger and colleagues to be highly accurate in an image segmentation task[111,112]. U-Net has been one of the most popular DL architectures in the bioimaging community and has since been adapted for various tasks in microscopy and bioimage analysis[113–116]. More recently, the success of another type of DL architecture, the generative adversarial networks (GAN)[117,118], has become increasingly popular for tasks in microscopy. GANs are trained by letting two networks, a generator, and a discriminator, compete against one another to generate a realistic image in the target domain and distinguish 'fake' from 'real' images. When the generator 'fools' the discriminator, it can be used to generate new images that appear extremely realistic in a certain domain, e.g. predicting a photograph from a drawing. GANs can be used for extremely diverse tasks. Unlike U-Net, they do not necessarily require paired images, which may have potential for tasks in which no paired images can be easily acquired (Fig. 3D).

How such architectures and learning strategies as discussed above are applied in microscopy challenges and bioimage analysis will be the subject of the following sections.

i. Denoising

As explained above, one of the major outstanding questions in microscopy is how to balance the dilemma of phototoxicity and resolution, with existing techniques approaching their technological or conceptual limitations. DL methods can address this question in different ways. A primary focus of DL on this is to improve on the existing plethora of denoising algorithms to enhance image data. As previously mentioned, existing denoising algorithms suffer from the problem of bias introduction by making assumptions about the imaging process and the type of noise in the image.

However, noise in an image can have multiple different components which can be difficult to model correctly in composite. This includes the noise inherent in all imaging systems such as shot noise and read noise. Shot noise is caused by the random arrival of photons on the detector at a given signal intensity, a result of light's particle-wave nature. The detection of these photons follows a Poisson distribution which for the amount of photons usually measured in imaging experiments is approximately normal[119,120]. The standard deviation of the signal due to shot noise therefore scales with the square root of the total number of detected photons, i.e. the square root of the signal's intensity. The contribution of shot noise to the standard deviation of the signal intensity thus becomes more significant where the signal intensity is small but is comparatively small at higher signal intensities. Read noise is caused by the random displacement of electrons in the detector for example due to the temperature of the detector itself (thermal noise), which is then translated to a pixel with non-zero intensity. The intensities of these pixels are usually Gaussian distributed[121], and unlike shot-noise are not dependent on the image signal. This means that even in the complete absence of light the detector will produce some pixels with a signal in the image, i.e. noise. The presence of these noise sources limits the dynamic range of images, specifically at small signal intensities where these types of noise can dominate, meaning that small differences in the signal intensity will not be distinguishable from noise. In microscopy, other sources can also change the total noise of the image. This includes for instance light from out-of-focus objects[122] or autofluorescence[123] or bleed-through from other fluorophores[124,125]. Furthermore, the contribution of each type of noise can differ between different samples or microscopes, which can make it more challenging to employ the same denoising method to differently acquired datasets.

Instead, several groups have attempted to remove noise from microscopic images by training CNNs, often with a U-Net architecture[1]. If trained successfully, no assumptions about the nature of the noise in the image are required, and noise based on specific imaging scenarios could be removed by training the CNN on a suitable dataset. One such application is content-aware image restoration (CARE) which uses a U-Net architecture and paired images (noisy and low noise) to learn denoising[113]. Paired images of different specimens were created by imaging fields-of-view (FOV) with high and low exposure or different exposure times and training CARE on this data. An application of this denoising approach is to image light-sensitive organisms under conditions with low phototoxicity, and denoising the resulting images using a CARE model trained on images acquired on fixed samples of this organism which can be done at higher light doses than live-specimens. Using CARE in this manner to denoise live-image data yielded high SNR images at greatly reduced light doses[113]. As expected, CARE also outperformed classic denoising algorithms in terms of the quality of the images produced. This was an early example of how DL could be exploited to address the crucial challenge of achieving strong signal at low phototoxicity in microscopy images, as outlined above. CARE was one of the first published DL-based denoising approaches for the bioimaging field which has since seen several similar approaches with slight differences in architecture[37,129].

A disadvantage of the supervised training approach used in CARE is the abovementioned dependency on paired training data which may be difficult to acquire under a microscope for some experiments, e.g. very dynamic systems. Hence, different groups have trained neural networks for denoising in an unsupervised manner[126–128]. The main assumption of unsupervised training for denoising is that image-noise, unlike image-signal, is pixel-wise independent, meaning that the contribution of noise to the intensity of one pixel does not depend on the intensities of any of the other pixels in the image. This was first exploited by Lehtinen et al. who showed that a network trained on pairs of two noisy images can learn the structure of the image noise while recognizing underlying structures[126]. The requirement of image pairs was removed entirely by further improvements by Batson et al. with the noise2self approach[127]. This demonstrated that removing random parameters from the input image and using an ANN to learn to exploit the remaining information to predict the removed information can outperform classic denoising approaches. A more specialised case of this approach was used by Noise2Void, which creates paired image patches where the input has one pixel removed[128], in a so-called blind-spot network. By training a U-Net-like blind-spot ANN to predict the missing pixel, the model exploits neighbouring pixel information to reconstruct underlying structures. Since noise should be pixel-wise independent, the model can theoretically only learn the correct pixel intensity by recognizing the signal of the underlying surrounding structure, rather than the noise. Such denoising approaches can reach surprising performance and the advantage of requiring no ground-truth information make them attractive for experiments where acquisition of low-noise ground-truth data is challenging, e.g. in timelapse acquisitions where it may not always be feasible to create paired images from the dynamic sample or to create a fixed training sample.

ii. Super-resolution

Beyond the ability to denoise images, supervised approaches can reconstruct super-resolution images from diffraction-limited images. For example, one of the original applications of CARE was the prediction of super-resolution radial fluctuation (SRRF) images from diffraction-limited images[113]. Similarly, ANN's are able to predict SRM images from conventional confocal microscopy images by using Stimulated Emission Depletion (STED) microscopy images as a high-resolution training dataset[130,131]. Using ANNs for super-resolution has been criticized within the bioimaging community, primarily because of the risk of undetected artifacts[80] . In CARE, it could also be argued that predictions may enhance details but do not achieve true super-resolution because predictions have the same number of pixels as inputs, thus not generating images with more information than input images.

Another advantage of DL for SRM is in single-molecule localisation microscopy. Although at least one approach has achieved SMLM resolution with a supervised paired image approach (ANNA-PALM)[37], an alternative route is to train algorithms to reconstruct images from the localisation data from images, essentially the same information used to reconstruct SMLM images with existing algorithms[130,131]. However, once trained on such data, DL algorithms have been shown to be faster and more efficient in reconstruction than conventional reconstruction methods. The latter point means, for instance, that fewer frames are needed to reconstruct an SMLM image or that the model allows for a higher emitter density than before. Using DL here can, therefore, make it faster, cheaper, and less toxic to perform super-resolution, addressing key shortcomings of existing SMLM reconstruction methods.

iii. Artificial labelling

In addition to improving existing software methods for microscopy and post-processing, DL offers exciting new avenues to tackle the challenges that persist in bioimaging. A prominent example of the innovative tasks DL can perform is artificial labelling[114,132]. This task attempts to label cells completely in silico, rather than introducing labels into cells. The rationale behind this approach is that it would allow cells to be imaged under brightfield conditions without the risk of inducing phototoxicity through ROS creation. Individual compartments could then be inferred from the brightfield image. Two groups initially demonstrated that this is possible in 2D and 3D and can be exploited from brightfield-to-fluorescence and EM-to-fluorescence images, inferring cellular structures, such as nuclear membrane, nucleoli, plasma membranes and mitochondria. In one of these, label-free prediction (fnet), it was also shown that successful translation between different image modalities could be achieved by training on a relatively small dataset of only 30-40 images. Artificial labelling has sometimes been used as a step in other DL pipelines. For example, in Lu et al., the prediction of labels is used to train a network to learn autonomously to separate different classes of objects in images[133].

A question that has yet to be convincingly answered by artificial labelling algorithms is whether they can detect features not present in a training dataset, i.e., discover truly ‘new biology’. Similar questions are sometimes seen as the main weakness of DL algorithms in general. Although this caveat must be acknowledged, DL-methods for artificial labelling may have applications beyond their ability for label prediction. As Lu et al. have shown, artificial labelling can be used as a step in other analysis pipelines, other than predicting structures. Furthermore, DL-methods that translate between modalities can be used to track nuclei without labelling cells[134]. It is also possible to use such methods for virtual tissue staining, by translating from a brightfield or fluorescence microscopy image to a histopathology-like sample[135] Such applications, which could be very beneficial for bioimage analysis, do not require ‘new biology’ to be detected while fulfilling the promise of reducing the risk of photodamage via fluorescence or the fixing of samples using toxic chemicals.

iv. Segmentation

A common task in bioimage analysis where DL’s powerful object detection capabilities can be exploited is object segmentation. There are two types of segmentation which are used in computer vision tasks, one of which is semantic segmentation the primary aim of which is distinguishing a foreground structure from background. This task consists primarily of separating objects of a specific class from the background, usually by creating a mask that outlines the edge of these objects. Semantic segmentation is used for example to identify large structures such as membranes in EM-data which stretch over significant areas of an image[111,115]. The other type of segmentation is instance segmentation which aims to distinguish individual objects by creating object masks around their edges. Instance segmentation is a common requirement for cell size measurements, cell counting, and tracking.

DL for semantic segmentation is already heavily employed in biomedical imaging as it requires significantly less effort than manual segmentation[55] and relatively less manual fine-tuning than existing segmentation methods[136–138]. Semantic segmentation usually requires inputs and target masks, which need to be manually or otherwise curated before training which is the most labour-intensive step in using DL for this purpose. However, after training, ANNs generally outperform classical approaches in accuracy and generalization[55,111,139–141], especially when performing cell segmentation in cocultures of multiple cell types[55]. This has led to wide use-cases in histopathology where sections can be segmented to detect cancer cells in tissues such as colon glands[142,143] and breast tissues[144,145].

One of the most common instance segmentation tasks in bioimage analysis is nuclear segmentation[146]. A difficulty in automating nuclear segmentation is separating overlapping or touching nuclei in a segmentation pipeline. This challenge in nuclear segmentation was specifically tackled by the StarDist method, which interprets masks of individual nuclei as polygons the edge points of which are used as training data for a U-Net architecture which learns to create the mask[147,148]. StarDist became one of the best nuclear segmentation algorithms, which more recent nuclear segmentation algorithms are only beginning to improve upon. By exploiting the architecture and backend of StarDist with a slight modification of the interpretation of the training masks, the SplineDist architecture has since been developed, which improves the versatility of the original StarDist and allows segmentation of shapes beyond the roughly circular shapes in the original StarDist[149]. This example also highlights how the open-source nature of some DL tools, such as StarDist which was documented and disseminated via GitHub, aids in the improvement and adaptation of algorithms by the community.

v. Object detection

From their inception, DL methods were designed to solve object classification problems in image data sets. The increased performance of novel methods is driven primarily by image analysis competitions that provide very large datasets, containing millions of images with thousands of classes, and ultimately rank the architectures that achieve the best performance in identifying the classes in the images[150–152]. It is often the highest ranked architectures, such as ResNet[65], VGGNet[153] or GoogleNet[154], which are then employed for research in tasks beyond the classes in the public image data for which these methods are developed. Bioimage analysts and histopathologists quickly identified the potential of using powerful classifier CNNs for microscopic image analysis. Compared to the often tedious process of manual annotation, such networks can automate classification while maintaining almost human-level accuracy[1]. This performance compares well with non-DL methods for classification, which rarely achieves human-like performance on complex datasets. Therefore, CNNs are already widely used in biomedical imaging, for example, to identify cancer cells in patient samples[155–158]. They have also shown promise in analysing high-content subcellular feature screens in drug studies[56,159,160].

An important aspect of using ANNs for classification tasks is their potential to use features in the image which may be very subtle for human annotators to identify. Thus, CNNs can affect the type of data acquired by microscopists. For example, CNNs, which do not require fluorescence markers to identify certain cell types, may free up the requirement of users to introduce labels into their specimens or to image them fluorescently. Indeed, CNNs for classification have been used for just such purposes in identifying dead cells[161], cell cycle stages[162] or stem-cell-derived endothelial cells[163].

The output of standard classification tasks is usually a simple class label predicted for a whole image. However, for many bioimaging tasks, this is not sufficient as images may contain hundreds of cells and the exact locations of the classified cells in the image need to be identified for analysis. The task of establishing the location and class of objects is known as object detection. Here, the object is often ‘detected’ by drawing a bounding box around the object. Image classification and object detection are similar in their goal, and it is common to exploit features learned in classification networks for object detection. To do this, one can use the trained weights of a classification model which can encode useful feature maps for a dataset, and add one or more layers which are trained to learn the correct location of bounding boxes via regression. Using powerful image classification architectures such as Resnet50, Resnet101[65] or EfficientNet[164] as a so-called backbone of such object detection methods is relatively common as these methods are well-tested and represent the state-of-the-art for classification tasks. They therefore promise to yield strong performance in object detection tasks as well. Thus, they are often the backbones of some of the best current object detection or instance segmentation architectures, which includes, for example, YOLOv2 (object detection)[94], Detectron[165] and MaskRCNN[166] (Detection and instance segmentation). To use these networks in biology, they must be retrained by transfer learning on a relevant data set and cannot be applied directly 'out-of-the-box'[102,167,168].

The main drawback of CNNs as classifiers or object detectors is the need to provide annotated training datasets. Firstly, this tends to require manual curation of large datasets with class labels and/or hand-drawn bounding boxes which may be a significant time investment. Secondly, there are currently several different formats for image annotation, and it can be difficult to use a specific method without the correct data format which may be difficult to create for certain data types. Finding a standardised format for image annotation or labelling remains an open problem when using existing CNNs for classification tasks.

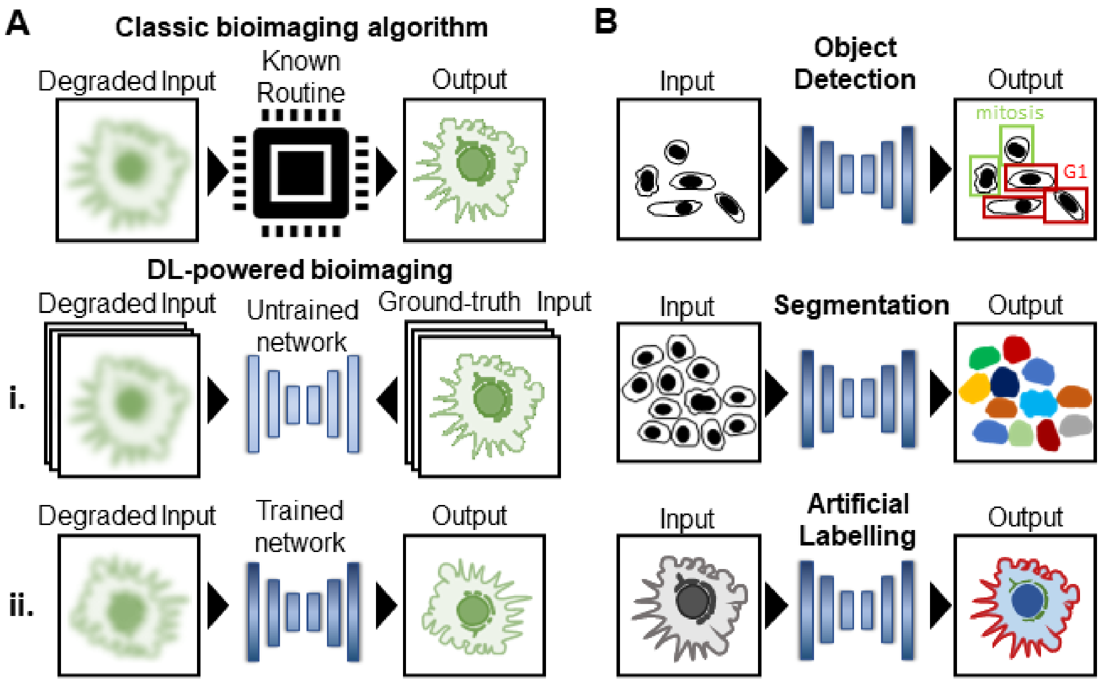


Figure 4 – Examples of DL tasks in bioimaging – A) *On top: classic algorithm for denoising/deblurring/resolution enhancement task. Below: i) ANN is trained on pairs of noisy-noise-free images and is used in ii) to denoise an unseen image. Once trained the network can be used in the same direction as the classic bioimaging algorithm.* ***B)*** *Three further examples of neural network tasks which would be trained the same as in A. (adapted from von Chamier et al., 2019[73]).*

I. 1. 3. On the benefit of democratisation of new technology for science

Besides biochemical approaches, microscopy remains the main tool that gives researchers access to the micro- and nano-cosmos. Therefore, much of our knowledge about cellular biology is connected to technologies that have enabled progress in microscopy and bioimage analysis. Hence, it is interesting to understand what features a successful technological innovation in microscopy must have to maximise its impact on cell biological research. One way in which this can at least qualitatively be defined in research is by assessing the rate of discoveries made by researchers. This can perhaps be further specified by considering how 'big' a discovery is, with a breakthrough regarded as having higher 'impact' than several incremental additions to knowledge. Although it may be impossible to quantify the 'impact' of discoveries, this approach may help to explain how technology influences research. Hence, a simple answer to the impact of technology on research is the extent to which it helps researchers make discoveries.

In the above exposition, several problems in microscopy and their current technology solutions are described. The impacts these technologies have had on biology differ significantly, but not only in terms of the nature of the questions they are trying to answer. While it may be argued that technology itself is transformative for science[169], this diminishes the role of the community that uses the technology. In the influential 'Laboratory Life' Bruno Latour observes that a lab's research output is intricately linked with the group's access to cutting-edge technology. Indeed, access to technology can even make or break a scientist's career prospects and is sometimes used as a gatekeeper to beat competitors[170]. In a competitive research environment, it is often the laboratories with access to the most novel tools that quickly impact their research areas. Although this initial impact of new technology is often stressed in retrospect, the true influence of a new technology on a research field is often not clear until it has reached most of its community. The technologies described above are all examples of this concept. In the first phase, they are developed by specialist groups with significant expertise and resources who want to solve a specific problem, e.g. the problem posed by the diffraction limit. In the next phase, the technology is assessed by other expert groups who might add features and replicate the technology. Usually, only after innovations have been assessed by the community of experts, are these tools adopted by the majority of users in the community who are not experts in the technology, but who can exploit it (Fig. 5). In this phase, tools such as SRM and bioimaging algorithms develop their full impact for the community, as they benefit from the full diversity of questions its users can ask and contribute to the creation of new knowledge on a wide scale. Hence, whenever novel technologies are developed there is an immediate drive to make them cheaper or simpler in use. This common practice within the scientific community will from hereon be referred to as 'democratisation'.

The impact of a technology can therefore be at least qualitatively assessed via the degree to which it becomes democratised. For example, the use of personal computers and hard drives with terabytes of storage has had a tremendous impact on the research performed daily by almost the entire bioimaging community (Fig. 5). In fact, the impact of the personal computer for microscopy is virtually impossible to assess because it has become the standard within a few decades. In contrast, SRM techniques and novel microscopy tools such as LLSM are difficult to use, expensive, less versatile, and therefore much harder to grow within the microscopy community. Yet, even SRM and LLSM have become significantly more accessible in recent years and a much larger proportion of users now has access to these tools than during the initial phase of their development, with many methods now commercialised or accessible via open-source resources[171,172] which will likely continue to grow their impact on biological research.

The phenomenon of technologies spreading in communities has been studied extensively in the social sciences and economics as 'diffusion of innovation'. Under this theory, Rogers identifies factors which can prevent a technology from being adopted by user-subgroups and thus fail to become ubiquitously used[173]. These include among others:

- the failure to convince groups of prospective users of the potential or purpose of an innovation,
- the complexity of the innovation, i.e. how easily and financially feasible it is to use the innovation.

Overcoming these obstacles is the role of democratisation. This also applies to the most recently introduced technological innovation in microscopy.

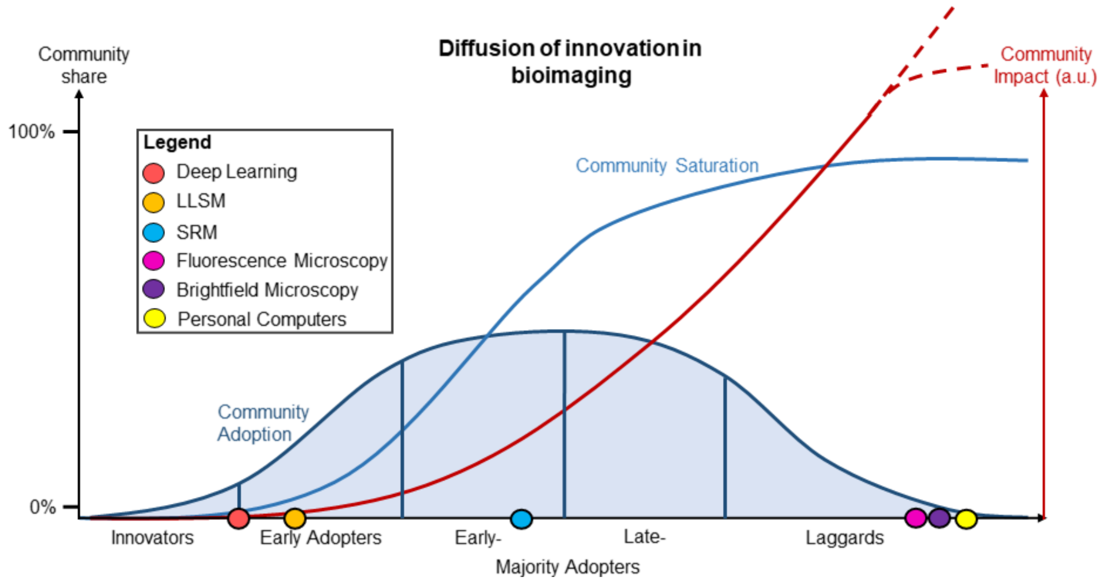


***Figure 5 - Diffusion of innovation in bioimaging** – Uptake of technological innovation goes through different phases (x axis). To enter each phase, barriers to further uptake need to be overcome (vertical blue lines). DL is in an early phase as many DL tools are still mostly distributed and used among developers (innovators) with some (early) adopters for specific tasks. Other tools have already progressed further in terms of their adoption and their impact. 'Impact' is added here (red) to symbolise how new technologies can continue to produce novel results for the community even after community saturation is reached. The exact impact as saturation is reached can be difficult to assess (dotted lines). The figure is adapted from the original proposal of diffusion of technology by Rogers, et al*[173].

When interpreting the advent of DL in microscopy from the perspective of the concepts of diffusion and democratisation, it likely sits between the first and second stages. However, some potentially ground-breaking techniques remain unused by most bioimaging communities and continue to be developed and used primarily by groups with significant expertise in DL. Taking three DL papers with big impact in the bioimaging community, those introducing CARE[113], StarDist[148] and fnet[114], and searching the papers citing these methods for use of the respective techniques in their research, it becomes clear that these tools, though widely cited by other method developers, were not yet widely used in the bioimaging community at the time of writing, though StarDist appears to have comparatively more users than the other methods (Fig. 6).

It can therefore be argued that despite the increase in the number of publications and their relative impact, in this case measured via citation, DL has currently not reached a level of uptake where it can develop its full potential for the research community, in contrast, for example, to image analysis tools such as ImageJ/Fiji. Phrased differently,

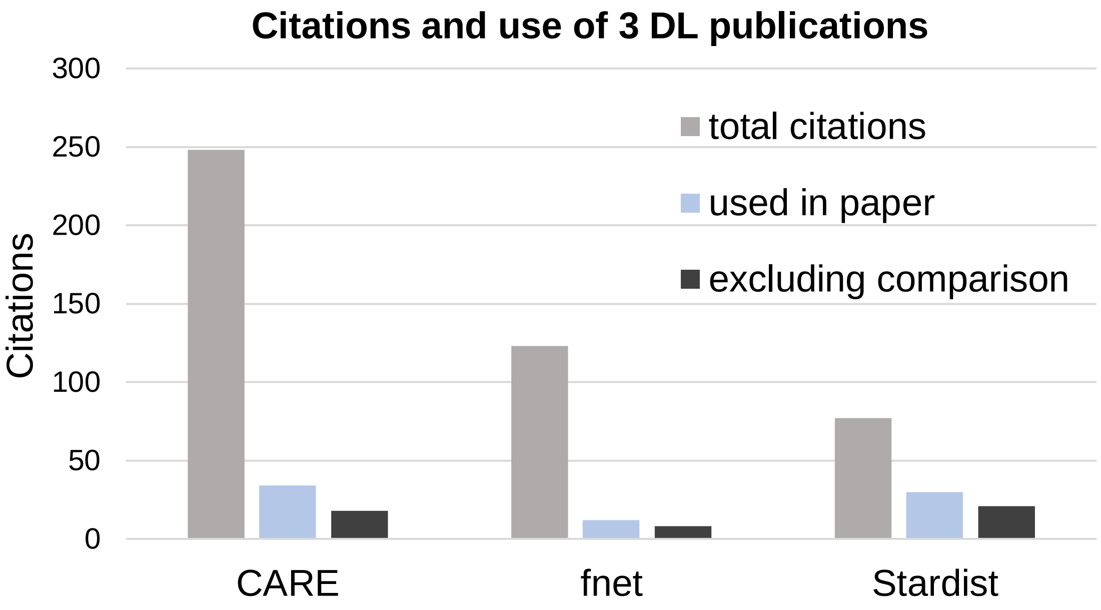


***Figure 6 - Citations and use of DL methods for bioimaging** – 'total citations' include reviews and any other citation found in a search using Google Scholar. ‘used in paper’ indicates the use of the tool in a paper which cites the original publication. ‘excluding comparisons’ includes bioimaging papers, rather than publications which only use the tool to compare their own tool with. (Results are based on citations up to 23rd March 2021).*

I. 1. 3. a. Barriers in the democratisation of DL for bioimaging

What are the reasons for the relatively low uptake of DL tools? Although the relatively small uptake of DL in the bioimaging community can be explained simply by the novelty of the technology, one can identify at least five reasons which are hampering the uptake of this specific technology in bioimaging which can be related to some of the obstacles which Rogers observed in his work on diffusion of innovation. I summarise these obstacles for DL in the bioimaging field as follows:

1. Dissemination problem – since DL is a comparatively novel technology, the existing tools are primarily disseminated between groups that develop DL tools and have a high degree of expertise in this field. Groups new to the field still often require the help of developers to utilise even some popular tools. This suggests that existing methods may often not be intuitive enough for novice users to independently employ on their own data.

2. Knowledge problem – This is directly related to the above point. Many biologists and even bioimage analysts are currently not familiar with DL and its jargon. Currently, users will often need at least an understanding of the underlying coding language, usually Python, to start using most published DL tools. While the number of users in the bioimaging community using Python packages, such as NumPy[174] and SciPy[175] is likely growing, using DL, and optimising tools for custom data requires additional knowledge of basic neural network concepts, which is still rare in the bioimaging community. Therefore, there has been a push within the community to give wider access to DL to novice users. Currently, the main strategy has been to provide access to trained models that can thus be applied by users. This includes U-Net[111,176], cDeep3m[177], DeepCell Kiosk[178], DeepMIB[179], NucleAIzer[180], YAPiC[181], ImJoy[182], the recent releases of ilastik[54] and CellProfiler[183] and the Noise2Void[128] and DenoiSeg[184] Fiji plugin. Furthermore, there has been an increase in interest in creating 'model zoos', which are collections of models trained on a wide variety of datasets that users can download and use locally on their data[182]. However, very few tools allow users to use DL easily for training, arguably the key to successfully using this technology. Achieving this will again often require a good understanding of Python and DL. There exists a need for easily accessible didactic DL tools for training novice users in the bioimaging community in this new technology and tools which do not require learning a new (code) language to use.

3. Hardware problem – DL is resource-intensive. DL tools are most valuable to users when they train neural networks on custom datasets. While it is possible to use models trained on other datasets for prediction, the performance of such models will almost always improve upon retraining on a dataset that represents the user's use case. If not, DL models are likely to produce suboptimal results or may even predict artefacts that are learned and incorrectly introduced from an unrelated dataset. Therefore, training is often essential for the successful use of a DL tool in a research project. However, this may not be simple for many users because DL training, specifically backpropagation and gradient descent, used with large datasets, will only be practically feasible if it is GPU accelerated. Setting up a DL workstation on-site would for most labs be a significant expense (1000s of pounds) and even setting up remote resources can incur significant costs. Furthermore, running a GPU workstation can be energy intensive, training a model on a Tesla P100 GPU for 4 hours can use up to 1kWh, consuming relatively large amounts of energy adding not only to the running costs of the lab but also increasing the environmental impact of the lab's research.

4. Reliability Problem – DL has limitations that somewhat set it apart from other tools commonly used in bioimage analysis, such as deconvolution or image reconstruction algorithms. Specifically, poorly evaluated neural networks can produce undetected artefacts and hallucinations[80]. Problematically, these hallucinations can be challenging to trace back through the complex architectures of neural networks, making the identification of the error-source challenging or even impossible. This has led to the characterisation of DL algorithms as 'black-boxes' which are challenging to analyse by users, and has led to scepticism towards DL tools in parts of the bioimaging community[185]. While such criticisms are often justified, the creation of artefacts in image reconstructions is not unique to DL[61]. Some of these can be addressed if DL tools are supplemented with concise and robust quality controls that allow users to identify problems quickly. Yet, such quality control steps may vary between tools and it is often up to the developers how these quality controls are implemented, how much guidance is given to users in establishing a quality control pipeline and how quality metrics can be interpreted. To improve uptake and trust in DL tools, quality control steps should be built into every pipeline and should be easy

and quick to use, readily interpretable and comparable between methods. This is currently not the case for the existing menagerie of DL tools.

5. Dataset problem – Many currently existing DL algorithms are based on supervised learning approaches and these approaches scale strongly with dataset size in terms of performance. Hence, many networks will only reach peak performance if they are trained on an appropriately sized dataset, often containing hundreds to tens of thousands of training pairs. Creating and curating such datasets for neural network training imposes a significant time and resource constraint on laboratories.

I. 2. Motivation of this project

As a new technology, DL is yet to enter the phase of maximal uptake in the community (Fig. 5) which is currently limited by the above obstacles. However, DL has massive potential for many significant problems for the bioimaging community, from challenges directly related to imaging (phototoxicity, resolution) and to image analysis (classification, segmentation). By the above reasoning, arguing for the benefit of democratisation of technologies to unleash their potential for discovery and thus science, it can be assumed that the democratisation of DL could be transformative for many researchers in the bioimaging community.

In this project, I have set out to create an avenue that could contribute to the democratisation of DL by addressing the limitations mentioned above in the field. In the following chapters, I will lay out how the above problems were addressed in the ZeroCostDL4Mic platform which is the main product of this research.

I. 3. Summary of main results

In the first chapter, I will explain how the problems outlined above influenced the creation of the platform and specifically how these considerations informed the platform's interface.

The main achievements of the work presented in chapter 1 are:

- Creation of the ZeroCostDL4Mic GUI, including a standardised workflow and detailed explanations of each step. This allows novice users to access DL without code interaction and gives access to GPUs through the exploitation of Google Colab – addressing problems 1, 2 and 3.
- By implementing different DL-methods, I can compare key features of the current DL-code and assess what characteristics make tools better for the bioimaging community. This gives a deeper understanding of Problems 1 and 2 and potential solutions.

The second chapter deals with the quality control problem in the DL tools currently used in bioimage analysis. I demonstrate how we implemented Quality Control (QC) in the platform created in chapter 1 and show how it can be applied in the platform to facilitate the use of DL in bioimaging.

The main achievements of the work presented in chapter 2 are:

- By applying standard quality metrics, I show that the ZeroCostDL4Mic platform can be used successfully as a powerful DL deployment tool, replicating many of the behaviours expected from ANNs. This means that despite its limitations, it can be used to train and deploy DL models which can produce meaningful results for research projects, and not only as a teaching or learning tool.
- I show that DL performance depends not only on dataset size, but also on the data type.
- Furthermore, I find limitations in the currently used quality metrics employed in image-based DL methods that may hamper their ability to determine DL performance.
- Integration of Quality Control is designed to directly address problem 4.

In the final chapter, I show an example of how the platform can be used for an emergent image analysis problem (unmixing fluorescent labels to free up imaging channels in microscopy).

The main achievements of the work presented in chapter 3 are:

- The tools included in ZeroCostDL4Mic allow users to easily explore datasets for their own analysis challenges, beyond those the implemented methods were developed for. This is important if the platform is to engage a broad variety of users and use-cases.
- The outputs from the notebooks give enough detail to thoroughly compare different methods and datasets, giving insights both into the DL methods and the datasets used to train them

Finally, I will discuss how the project can inform the integration of future DL tools into bioimaging, and what specific aspects of implementation should be considered. This includes insights gained about the use of quality metrics and the types of data sets for which DL can be used effectively and whether there are domains in cell biological imaging data for which DL solutions may currently remain unsuitable.

II. Results Chapter 1 – Creating the ZeroCostDL4Mic Platform

II. 1. Overview of ZeroCostDL4Mic

The key aim of this project is to accelerate the uptake of DL tools by the microscopy community. As a novel technology, its democratisation can fuel new research and lead to discoveries. In the Introduction, I identified 5 problems for the democratisation of DL technology:

1. Dissemination Problem
2. Knowledge Problem
3. Hardware Problem
4. Reliability Problem
5. Dataset Problem

Recognizing these obstacles and the benefits to the bioimaging community in overcoming them was the inspiration for the development of the ZeroCostDL4Mic platform. In this chapter, I will describe how this platform was developed, starting with the creation of the standardised DL workflow and its step-by-step integration into a graphical user interface (GUI). However, it is helpful to describe briefly what ZeroCostDL4Mic is and what it achieves to understand the significant efforts required to create it. Therefore, before explaining the details of its creation, I will briefly outline the ZeroCostDL4Mic platform. For full details, see our publication [[134]].

ZeroCostDL4Mic is a repository of notebooks created in Google Colab, which, respectively, implement a published DL method for applications in bioimage analysis. Its key aim is to make these methods more accessible to various users in the bioimaging community and DL-novices.

Colab has several features that make it an attractive solution for problems related to the accessibility of DL tools. The most important one is that it provides free access to GPUs, which is essential for training neural networks, as discussed in the introduction. This allows users to fit DL-methods to their data, without investing financial and time resources into procuring GPUs or GPU time. Colab notebooks follow the format of the widely used Jupyter[186] notebook for Python methods, but come equipped with hundreds of software packages pre-installed and are available through a web-browser, i.e. require no installation on a local machine. Like Jupyter notebooks, Colab notebooks can contain text cells to introduce, explain, and annotate sections of code. The code itself is placed in code cells that execute a block of code when initiated by the user by pressing the play button of the code cell, in the notebook interface. In contrast to Jupyter notebooks, ZeroCostDL4Mic exploits a feature specific to Colab, which is its ability to hide the code in code cells behind text or interactive graphical modules (Fig. 1) by using the Colab *forms* modules. This means that all ZeroCostDL4Mic notebooks, in their default state, are entirely text- and image-based, thus allowing users to use any implemented DL method on the platform without code interaction. This addresses the important obstacle of accessibility because the backend code runs smoothly and intuitively. The code-free implementation of different DL-methods is a key achievement of this project, but, as I will explain below, it was also one of the central challenges during its development.

To appeal to a wide user base in the community, the methods we implemented in ZeroCostDL4Mic primarily address current challenges in bioimaging. The core methods released in ZeroCostDL4Mic are for denoising (CARE 2D/3D[113], Noise2Void 2D/3D[128]), super-resolution (Deep-STORM[131]), image segmentation (U-net 2D/3D[111,115], StarDist 2D/3D[147,187], SplineDist[149], CellProfiler[183]), object detection (YOLOv2[94]) and artificial labelling (label-free prediction/fnet 2D/3D[114]). In addition, we implemented methods which at the time of ZeroCostDL4Mic's creation had few applications in bioimaging[188,189] but which are so versatile that many applications are conceivable. These are the pix2pix[117] and cycleGAN[118] methods, which use generative adversarial networks and can, in theory, translate images between any two modalities.

The ZeroCostDL4Mic workflow and GUI are agnostic to the underlying methods, which means that all notebooks have an almost identical appearance and can be navigated in the same way by users. It allows methods beyond the core ones described above to be brought into the same format. In fact, this has already occurred, and the repository continues to grow (https://github.com/HenriquesLab/ZeroCostDL4Mic).

To summarise, ZeroCostDL4Mic addresses several of the problems identified in the previous chapter. Colab removes the financial and technical constraints of gaining access to powerful hardware, though with some limitations in terms of compute time and storage (see II.2.4). The GUI built around existing methods provides code-free access to powerful DL-tools with many uses in bioimaging. Furthermore, the workflow steps and annotations are specifically designed to educate users and promote DL's best practices, which will aid in increasing the appeal and user base of DL in bioimaging tasks to improve their democratisation.

The development of this platform required the review and use of many different tools and their adaptation to the project's desired format. In this chapter, I will outline how the central challenge in this project was solved: creating a consistent and accessible pipeline for a wide range of tools, without code access. The chapter follows the structure of the ZeroCostDL4Mic workflow. In each section, I first introduce the steps' objectives, which were always designed with the goals of simplicity and accessibility, and then outline how and how well these aims were ultimately achieved in the ZeroCostDL4Mic notebooks. To avoid redundancies, since several methods required very similar implementations, in this chapter, I focus on only 4 different notebooks, which represent the different main types of tools encountered in this project and cover the whole spectrum in terms of their ease-of-adaptation. These notebooks are for the CARE, U-Net 2D, fnet, and YOLOv2 methods. Furthermore, since this project was developed in collaboration with Romain Laine and Guillaume Jacquemet, these were the methods in which I was the only (YOLOv2, fnet) or a major (CARE, U-Net 2D) contributor. Therefore, for all steps with significant differences between these methods and their implementation, I treat them separately. Showing how these methods were implemented provides enough detail to understand how other notebooks in the project were developed and how further methods could be adapted in the future.

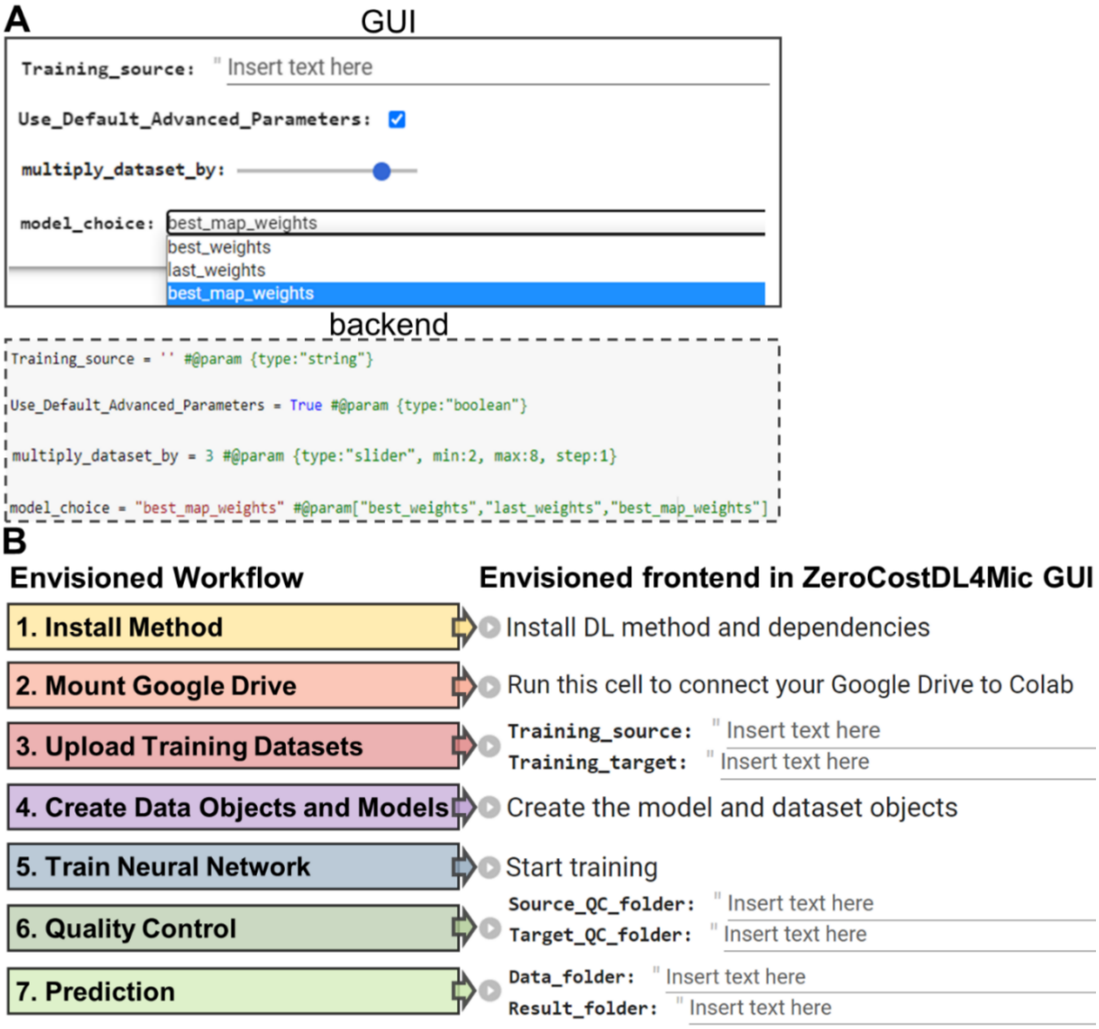
In the final section of this chapter, I will summarise the key achievements and discuss the insights gained from adopting a wide range of methods for use in ZeroCostDL4Mic. I will discuss which specific barriers in the code of current DL-methods were identified and how these influenced the creation of the presented platform. Additionally, it is instructive beyond this project, as future DL tools could incorporate the features that I identify here as crucial for the accessibility of DL for bioimage analysis.

II. 2. Building the ZeroCostDL4Mic notebooks step-by-step

II. 2. 1. Core DL-workflow for bioimaging tasks

Unlike conventional image analysis algorithms, DL workflows usually require significant user interaction before being used for custom tasks. The central task within this workflow is the training step that determines the model's performance for the analysis tasks. However, preceding a model's training, users will have to curate a training dataset, bring the dataset into a format suitable for the model to be trained, e.g. adjusting the size or datatype of the images, and choose hyperparameters for training. Different DL applications vary significantly in how the steps preceding, following, and including training are implemented in code. In some cases, these steps can be challenging to perform and often require the user to understand the developer code, which can be a challenging task even for experienced scriptwriters.

The variety in how developers implement the steps of DL workflows makes it challenging to apply the knowledge acquired in one DL tool to a different one, a potential source of confusion for users unfamiliar with DL. We thus set out to standardise the workflow steps of DL tasks so that methods for such different tasks as denoising and object detection can be performed by a user using the same steps. After deciding on the core methods to be implemented in the ZeroCostDL4Mic platform, we established a standard workflow that all notebooks should follow which contains 7 steps: Installation, Google drive mounting, uploading datasets, creating data and model objects, executing training, performing quality control on trained models, and using models for batch processing on unseen data. If datasets are already uploaded to the user’s google drive, this pipeline contains only 6 separate steps the user will usually engage with. The pipeline and its appearance in the ZeroCostDL4Mic GUI are shown in Fig. 7.


A
GUI
backend
B
Envisioned Workflow
Envisioned frontend in ZeroCostDL4Mic GUI
1. Install Method
2. Mount Google Drive
3. Upload Training Datasets
4. Create Data Objects and Models
5. Train Neural Network
6. Quality Control
7. Prediction
Install DL method and dependencies
Run this cell to connect your Google Drive to Colab
Create the model and dataset objects
Start training

Training_source: " Insert text here

Use_Default_Advanced_Parameters: ☑

multiply_dataset_by:

model_choice: best_map_weights
best_weights
last_weights
best_map_weights

```
Training_source = '' #@param {type:"string"}

Use_Default_Advanced_Parameters = True #@param {type:"boolean"}

multiply_dataset_by = 3 #@param {type:"slider", min:2, max:8, step:1}

model_choice = "best_map_weights" #@param["best_weights","last_weights","best_map_weights"]
```

Training_source: " Insert text here
Training_target: " Insert text here

Source_QC_folder: " Insert text here
Target_QC_folder: " Insert text here

Data_folder: " Insert text here
Result_folder: " Insert text here

Figure 7 - The ZeroCostDL4Mic GUI and workflow – ***A)*** *Shown are the four primary interactive features used in the ZeroCostDL4Mic GUI (from top to bottom: Text input field, tick box, slider and drop-down choices) to allow users to input parameters without code interaction and how this is implemented in the backend code of the notebook, using the forms module.* ***B)*** *(Left) The intended standardized workflow. (Right) How the GUI is intended to incorporate the workflow steps, with user entering paths and playing cells in a code-free environment.*

In Colab, each step is separated into a distinct code block that is played individually by the user. Instead of running the whole script at once, this separation allows the user to engage carefully with each step of the pipeline, which aims to help them familiarise themselves with the workflow and best practices of the DL pipeline.
After the workflow steps and the idea of using a code-free interface were established, I could begin the implementation of different DL methods in ZeroCostDL4Mic. This presented the central challenge of this project and is best summarized by the question: How can a standard DL workflow be created with the same user inputs while the underlying methods perform DL very differently? And how can this be achieved without significantly changing the methods themselves?

The effort to find these solutions varied significantly among the DL methods implemented and often depended on how the authors of the method implemented their specific DL tool. This directly influenced the ease with which different methods could be adapted for this project. In the following sections, I will show how we found solutions to implement all the DL methods chosen for ZeroCostDL4Mic for each step in the workflow. As I explained above, I focus on a subset of methods, firstly because these are the methods that I was primarily involved in developing for this project. These represent all the main types of backends encountered in this project. This includes CARE, which uses the CSBDeep type packages (also used in StarDist and Noise2Void), U-Net which is a Keras based method which is entirely created within the ZeroCostDL4Mic notebook (similar to Deep-STORM), YOLOv2 which is also based on Keras and uses separate configuration files, and fnet which uses a PyTorch[190] backend and required several custom functions to be writing in the notebook to be implemented (similar to pix2pix and cycleGAN). By focusing on these exemplary methods, I avoid showing redundant implementation aspects that would be nearly identical for methods of the same ‘type’.

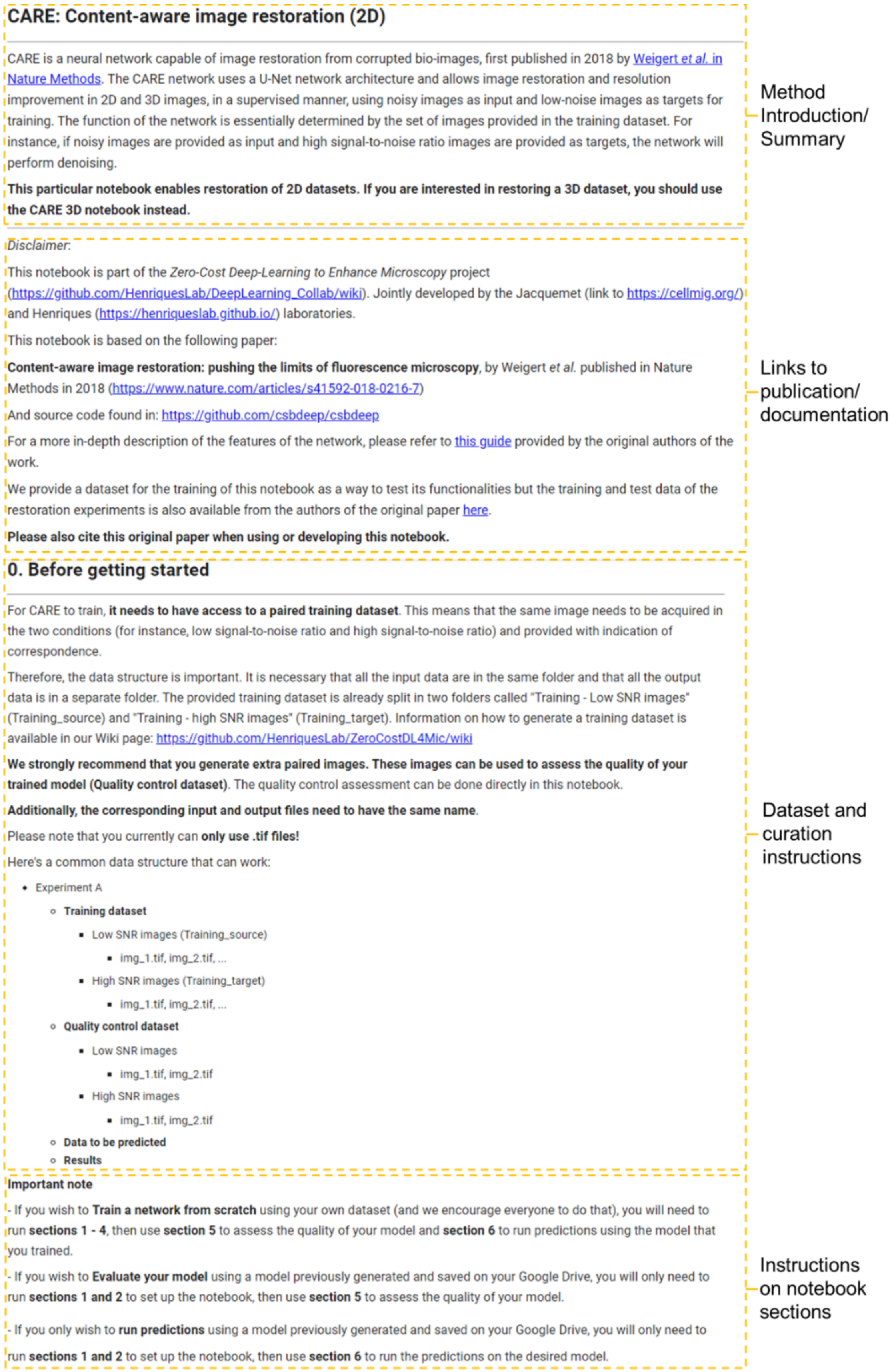
II. 2. 1. a. Method Documentation

Aims: To use a DL-method, the user needs to know quickly what it is for and what they need, beyond the notebook, to use it. Therefore, each notebook should briefly explain the purpose of the method and specifically the exact data type they need and how it should be structured to use the notebook without errors. In addition, the user must reference the publication, authors, or resource from which the notebook is adapted.

Each ZeroCostDL4Mic notebook begins with a text section that explains the purpose of the notebook and the authors of the original code (Fig. 8). Next, the user is guided through the types of datasets they need to provide. This differs between methods. While methods created for bioimaging purposes (e.g. CARE, U-Net, fnet) generally require .tif files, either as single-channel 2D images or as .tif stacks, others (e.g. YOLOv2) were designed for .png files. Furthermore, we give additional details about the datasets with which the notebooks were tested, such as their dimensions, number, and bit-depth. These values can guide the user to properly curate their dataset for the notebook.

The section on data type also contains instructions about the nomenclature conventions and folder structure of the datasets, which is usually necessary for supervised learning to construct training pairs of images. By default, developers choose different ways to achieve this. For example, in fnet, training pairs could theoretically be contained in any folder under any name, but training pairs need to have their paths stored in a path .csv file under columns named *input* and *target.* Most other methods require that the data pairs be at least in the same parent folder and require that the images in a training pair have identical names. For the ZeroCostDL4Mic notebooks, we chose this type of naming convention as the standard method, as we deem it instructive for users to engage carefully in dataset curation and encourage users to rename and structure their datasets in the same way for all notebooks, including those which by default do not require this from users.

The outer appearance of the introductory cells is identical in terms of formatting between all notebooks and is designed to give a sense of familiarity once a user has used any of the notebooks and switches to a different one. This should promote users' confidence in their ability to use different notebooks and quickly understand the steps necessary to curate their datasets and begin training neural networks. The standard layout is also useable as a template for new methods developed using the ZeroCostDL4Mic GUI.



CARE: Content-aware image restoration (2D)

CARE is a neural network capable of image restoration from corrupted bio-images, first published in 2018 by Weigert *et al.* in Nature Methods. The CARE network uses a U-Net network architecture and allows image restoration and resolution improvement in 2D and 3D images, in a supervised manner, using noisy images as input and low-noise images as targets for training. The function of the network is essentially determined by the set of images provided in the training dataset. For instance, if noisy images are provided as input and high signal-to-noise ratio images are provided as targets, the network will perform denoising.

This particular notebook enables restoration of 2D datasets. If you are interested in restoring a 3D dataset, you should use the CARE 3D notebook instead.

Disclaimer:

This notebook is part of the *Zero-Cost Deep-Learning to Enhance Microscopy* project (https://github.com/HenriquesLab/DeepLearning_Collab/wiki). Jointly developed by the Jacquemet (link to https://cellmig.org/) and Henriques (https://henriqueslab.github.io/) laboratories.

This notebook is based on the following paper:

Content-aware image restoration: pushing the limits of fluorescence microscopy, by Weigert *et al.* published in Nature Methods in 2018 (https://www.nature.com/articles/s41592-018-0216-7)

And source code found in: https://github.com/csbdeep/csbdeep

For a more in-depth description of the features of the network, please refer to this guide provided by the original authors of the work.

We provide a dataset for the training of this notebook as a way to test its functionalities but the training and test data of the restoration experiments is also available from the authors of the original paper here.

Please also cite this original paper when using or developing this notebook.

0. Before getting started

For CARE to train, **it needs to have access to a paired training dataset**. This means that the same image needs to be acquired in the two conditions (for instance, low signal-to-noise ratio and high signal-to-noise ratio) and provided with indication of correspondence.

Therefore, the data structure is important. It is necessary that all the input data are in the same folder and that all the output data is in a separate folder. The provided training dataset is already split in two folders called "Training - Low SNR images" (Training_source) and "Training - high SNR images" (Training_target). Information on how to generate a training dataset is available in our Wiki page: https://github.com/HenriquesLab/ZeroCostDL4Mic/wiki

We strongly recommend that you generate extra paired images. These images can be used to assess the quality of your trained model (Quality control dataset). The quality control assessment can be done directly in this notebook.

Additionally, the corresponding input and output files need to have the same name.

Please note that you currently can **only use .tif files!**

Here's a common data structure that can work:

- Experiment A
  - **Training dataset**
    - Low SNR images (Training_source)
      - img_1.tif, img_2.tif, ...
    - High SNR images (Training_target)
      - img_1.tif, img_2.tif, ...
  - **Quality control dataset**
    - Low SNR images
      - img_1.tif, img_2.tif
    - High SNR images
      - img_1.tif, img_2.tif
  - **Data to be predicted**
  - **Results**

Important note

- If you wish to **Train a network from scratch** using your own dataset (and we encourage everyone to do that), you will need to run **sections 1 - 4**, then use **section 5** to assess the quality of your model and **section 6** to run predictions using the model that you trained.

- If you wish to **Evaluate your model** using a model previously generated and saved on your Google Drive, you will only need to run **sections 1 and 2** to set up the notebook, then use **section 5** to assess the quality of your model.

- If you only wish to **run predictions** using a model previously generated and saved on your Google Drive, you will only need to run **sections 1 and 2** to set up the notebook, then use **section 6** to run the predictions on the desired model.

Method Introduction/ Summary
Links to publication/ documentation
Dataset and curation instructions
Instructions on notebook sections

Figure 8 - Documentation of the ZeroCostDL4Mic notebooks *- Summary of the key sections in the introduction of the notebook for the example of CARE. This set-up is identical for all ZeroCostDL4Mic notebooks.*

II. 2. 1. b. Installation of DL-methods

Aims: In this step, the DL method is installed in the Colab environment. After playing the cell all the packages and functions needed by the DL method or elsewhere in the notebook should be installed. This ensures that all functions used in the notebook are available, no matter which cell is played after the installation cell. This is necessary so that cells later in the workflow do not depend on cells higher up in the workflow and can be played independently.

i. CARE 2D

Like other CSBDeep methods, CARE is directly installed as a Python package in the */usr/local/lib/python3.7/dist-packages* directory in Colab, with which users will generally not interact. Any methods necessary for CARE can be called after installation via its custom functions.

ii. U-Net 2D

Some published tools are built using basic Keras, a high-level language using TensorFlow[191] as a backend. Keras is native to the Colab environment and requires little effort to integrate into the notebooks. In the case of U-Net 2D, it was possible to create a function in the installation cell using Keras functions. This means that the functions defining the network architecture, building the training dataset objects, and defining the execution of model training and prediction could be defined here. For this purpose, the code for U-Net was simply transferred from a public repository into the installation cell, where any adjustments could be made easily. In the later cells of the notebooks, including those discussed below, this means user input could be simply used as input arguments for the functions defined in this cell, simplifying the implementation.

iii. YOLOv2

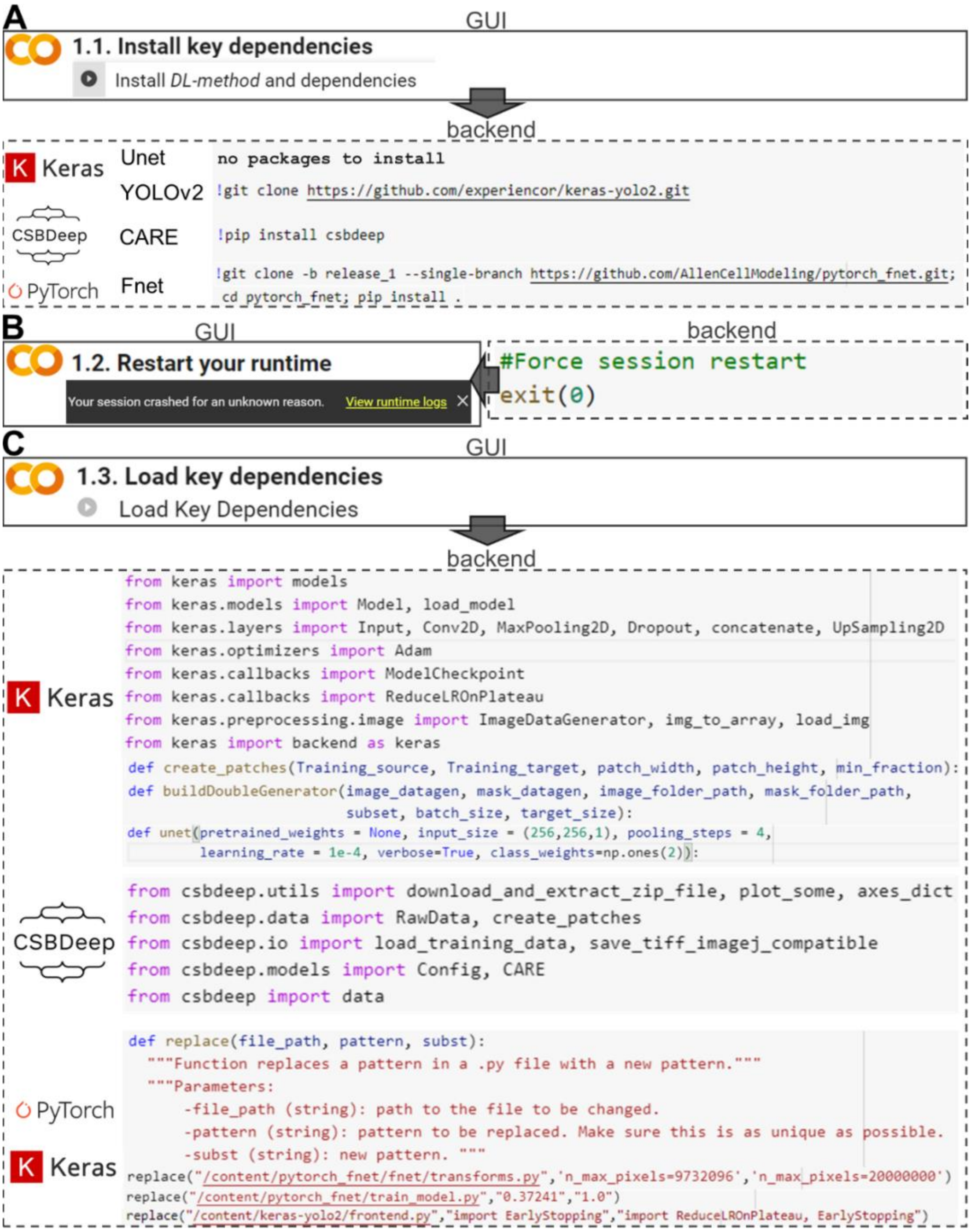
The YOLOv2 implementation in ZeroCostDL4Mic also uses Keras as a backend, but the functions used to build the model, dataset objects, training, and other functions are complex and in their public repository are defined on many separate Python scripts. Transferring these functions into the installation cell would have been significantly more challenging than for the U-Net notebook and would have been difficult to navigate for users. Instead, the YOLOv2 method is cloned from its GitHub repository into the Colab environment via the /content folder. From here, all YOLO functions can usually be accessed in the notebook after a Python import. The /content folder of the Colab notebooks created in each new runtime disappears at the end of the runtime. Although this means that changes to files in the content folder cloned from GitHub are lost when the notebook is restarted, this prevents files from occupying space on the users' drive, thus frees space for data that can be used for training. Compared to building methods entirely in the notebook, accessing functions via GitHub is less flexible and changes to the source code, often require editing files 'on the fly'. For example, originally, the YOLOv2 model training function does not contain learning rate decay. However, a decaying learning rate can help models converge to minima as the slope of the loss function decreases and is commonly used in DL[76]. However, including this in the training function in YOLOv2 required adding multiple lines of code to the function that builds the model in a script file. To facilitate this editing without user interaction, I created a function that overwrites the document and adds the lines of code if they are missing. The new training function can now be used with learning rate decay (Fig. 3 bottom). Similar adjustments were made in other functions in the YOLO notebook (not shown).

iv. fnet

fnet uses PyTorch as a backend, another Python-based deep learning package that currently competes with TensorFlow based methods for primacy in the DL field. Pytorch is not natively supported in the Colab notebooks. Hence, fnet, and other PyTorch based methods, need to be installed together with all necessary PyTorch packages, specifically, compatible CUDA packages, which ensure smooth performance on Colab's GPUs. Similar to YOLOv2, the fnet method is too complex to be feasibly transferred directly into the Colab notebook. Therefore, any changes to the fnet functions, such as user inputs, need to be edited in script files or Python scripts defining the training functions. Therefore, the installation cell in fnet contains replacement functions similar to those of YOLOv2.

Since many DL-methods are not as regularly updated by developers as Colab, there is a moderate risk that package versions included in the Colab environment such as NumPy or matplotlib are more recent version than those which the respective DL methods require. Therefore, it was necessary in almost all notebooks to replace Colab native packages by uninstalling the incompatible and installing the compatible version of these packages. However, to update the packages, the notebook runtime needs to be restarted, which usually requires user interaction and would be a confusing step for a novice. Hence, to avoid users having to restart the Colab runtime manually, the notebook is crashed in a controlled way to restart the runtime and update the required packages, without user input. After the crash, all packages, including any re-installed packages, can be imported into the runtime via import.

Due to this requirement, the installation section in the notebooks is three-tiered. The first cell (1.1.) installs all the packages that are not native to Colab; this usually includes the respective DL-package or repository, CUDA and torch packages, and any required versions for compatibility with the DL method. After installation, the notebook crashes and is automatically restarted, without user interaction. This step is denoted by a text cell to explain to users why the notebook crashed. Finally, packages are imported into the runtime, and any additional Python functions are defined. The frontend of the GUI and some typical features of the backends are shown in Fig. 9.



1.1. Install key dependencies

Install *DL-method* and dependencies

```
no packages to install
!git clone https://github.com/experiencor/keras-yolo2.git
!pip install csbdeep
!git clone -b release_1 --single-branch https://github.com/AllenCellModeling/pytorch_fnet.git;
 cd pytorch_fnet; pip install .
```

1.2. Restart your runtime

Your session crashed for an unknown reason. View runtime logs

```
#Force session restart
exit(0)
```

1.3. Load key dependencies

Load Key Dependencies

```
from keras import models
from keras.models import Model, load_model
from keras.layers import Input, Conv2D, MaxPooling2D, Dropout, concatenate, UpSampling2D
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
from keras.callbacks import ReduceLROnPlateau
from keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
from keras import backend as keras
def create_patches(Training_source, Training_target, patch_width, patch_height, min_fraction):
def buildDoubleGenerator(image_datagen, mask_datagen, image_folder_path, mask_folder_path,
                         subset, batch_size, target_size):
def unet(pretrained_weights = None, input_size = (256,256,1), pooling_steps = 4,
         learning_rate = 1e-4, verbose=True, class_weights=np.ones(2)):
```

```
from csbdeep.utils import download_and_extract_zip_file, plot_some, axes_dict
from csbdeep.data import RawData, create_patches
from csbdeep.io import load_training_data, save_tiff_imagej_compatible
from csbdeep.models import Config, CARE
from csbdeep import data
```

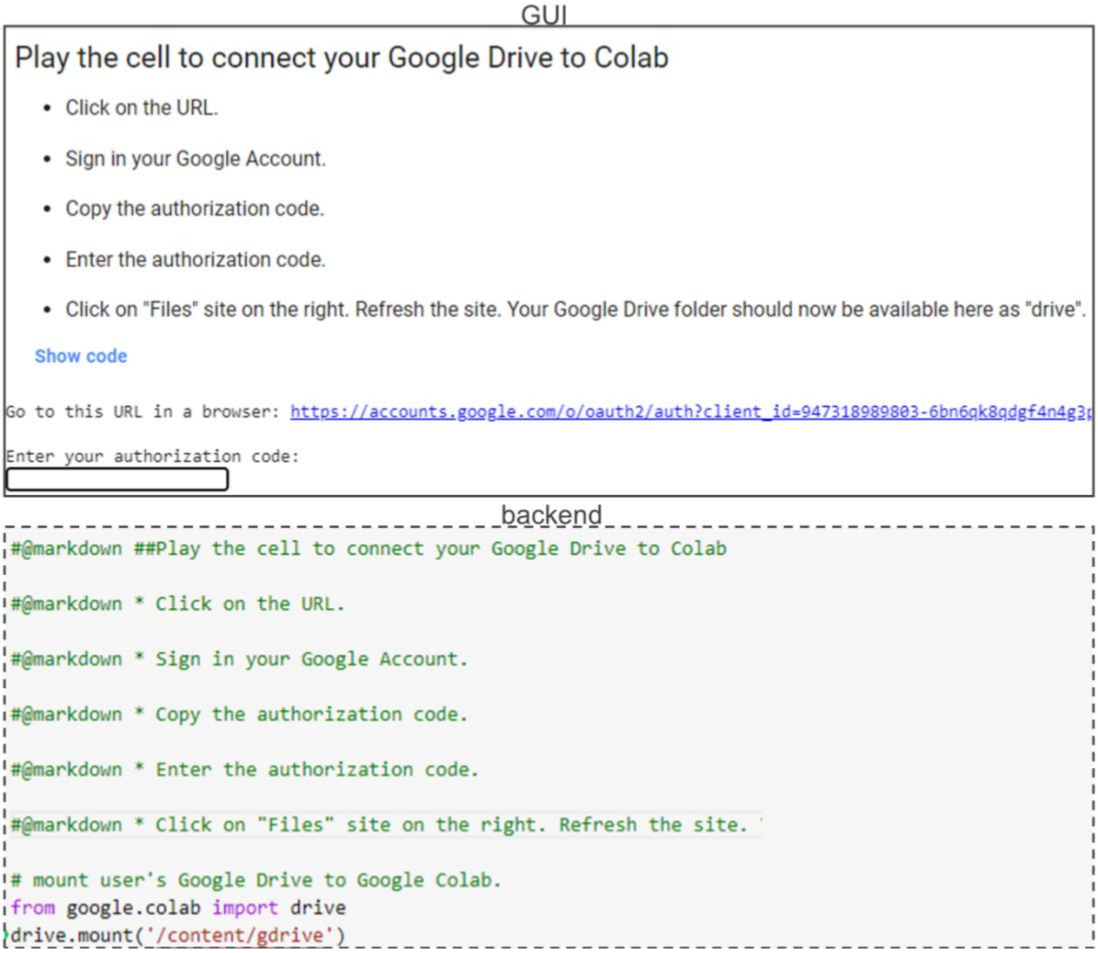
```
def replace(file_path, pattern, subst):
  """Function replaces a pattern in a .py file with a new pattern."""
  """Parameters:
      -file_path (string): path to the file to be changed.
      -pattern (string): pattern to be replaced. Make sure this is as unique as possible.
      -subst (string): new pattern. """
replace("/content/pytorch_fnet/fnet/transforms.py",'n_max_pixels=9732096','n_max_pixels=20000000')
replace("/content/pytorch_fnet/train_model.py","0.37241","1.0")
replace("/content/keras-yolo2/frontend.py","import EarlyStopping","import ReduceLROnPlateau, EarlyStopping")
```

***Figure 9 - Step 1 – Installing different DL-methods into the Colab environment - A)** The first cell installs the DL methods if they are not native to Colab. Only methods built entirely (U-Net) in the notebook require no methods installed in the notebook, via cloning repositories from* GitHub *or by installing* P*ython packages. The frontend of the GUI looks identical for all different DL or coding backends. (Other functions installed to ensure that the methods run correctly in the notebook are not shown here for simplicity)* ***B)** The Installation cell crashes the notebook, via the command shown on the right. This is explained to the user in a separate cell which requires no further interaction.* ***C)** Any methods or functions used in the notebook are installed in this section. For* K*eras-based methods, functions and classes are imported from the native libraries in Colab. Any functions required for the notebook can be defined here and can use parameters which are convenient for implementation in the notebook (e.g. Training_source, Training_target etc.). The functions shown here are only an example of the full code in the notebook. Methods from the CSBDeep project, e.g. CARE, can be imported from the previously installed* P*ython package (see A) which is sufficient to run the notebook. In methods such as fnet, are not generally imported into the notebooks but use script files in the repository. Hence, any changes in parameters, such as the number of pixels or specific numerical values can be inserted into these functions via the replace function, written for the notebook. The function ensures that if the 'pattern' argument is not found, the functions remain unchanged, ensuring that functions are not repeatedly edited. The approach used for YOLO uses a combination of all methods where some methods are imported, some are imported from the repository, and some are via 'replace'.*

II. 2. 1. c. Mounting the Google Drive

Aims: This section aims to enable users to easily get access to their datasets in the notebook with minimal effort.

To use Colab for customised tasks, on datasets provided by users, we determined the most efficient solution to be to access user data via Google Drive. Alternatively, datasets can be loaded into the Colab session directly via a Colab specific load function. However, this solution is inconvenient as it is prolonged and depends on the internet connection bandwidth. Uploading files in this way would also need to be repeated each time the runtime is restarted. Another alternative is loading files from GitHub repository and cloning the repository into Colab. However, this would require the user to already have or need to set up a GitHub account. This would run counter to the goal of providing the most straightforward possible access to the platform. Google Drive is a cloud storage platform that is already widely used and was therefore deemed suitable for most users. In this notebook step, the user plays a cell named '*Play the cell to connect your Google Drive to Colab'* (Fig. 10). In the backend, the cell uses a Colab command to open a hyperlink and an input field. The user follows the hyperlink and the instructions to create an access code entered in the input field. This will connect the user's google drive with all included files and folders to the notebook. To use files in a respective DL-method, the user can now access any datasets contained in the Google drive by navigating on a menu on the left-hand side of the notebook, which is saved in Colab's */content* folder, which is the default directory established for every Colab runtime.



Play the cell to connect your Google Drive to Colab

- Click on the URL.
- Sign in your Google Account.
- Copy the authorization code.
- Enter the authorization code.
- Click on "Files" site on the right. Refresh the site. Your Google Drive folder should now be available here as "drive".

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3

Enter your authorization code:
```

```
#@markdown ##Play the cell to connect your Google Drive to Colab

#@markdown * Click on the URL.

#@markdown * Sign in your Google Account.

#@markdown * Copy the authorization code.

#@markdown * Enter the authorization code.

#@markdown * Click on "Files" site on the right. Refresh the site.

# mount user's Google Drive to Google Colab.
from google.colab import drive
drive.mount('/content/gdrive')
```

Figure 10 - Step 2 – Connecting to a Google Drive - *In the GUI, the user plays the cell which opens the authorization code dialogue. By following the instructions, the user generates an authorization code which they can enter into the dialogue and mount their google drive to the Colab notebook. The lower cell shows how this is implemented in the notebook and that the Google drive will be available as gdrive in the notebook's '/content' folder. Note that the exact steps are subject to change due to updates in how Google Colab accesses users' Google Drive.*

II. 2. 1. d. Entering Training paths and hyperparameters

Aims: Before training a neural network on a custom dataset, the data users will usually curate for training needs to be converted into a DL compatible format, which means a Python object that is compatible with the training function of the respective method. Similarly, a neural network needs to be instantiated as a Python object and several parameters, such as the validation split, need to be defined by the user to set up the training process. Combining these steps into one section would allow users to see almost all the necessary parameters and variables necessary for training in a single cell. The goal is to allow user to enter the paths to an input and target dataset and the training parameters into GUI, which should be sufficient to create the dataset and model objects compatible with the respective DL method (Fig. 11). The user should also choose a name for their model. Under this name, we create a folder where all relevant files, including the model weights, for the trained model can be accessed later.

Since the platform is aimed to be conducive to users' experimentation with DL models, we also want to ensure that the training parameters and the most important aspects of the training dataset (i.e. number of elements, size) are saved in a readable format, allowing the user to review or replicate the training on the model. Finally, the user should be able to see an example of their dataset before the next step to ensure that the data was correctly loaded into the notebook.

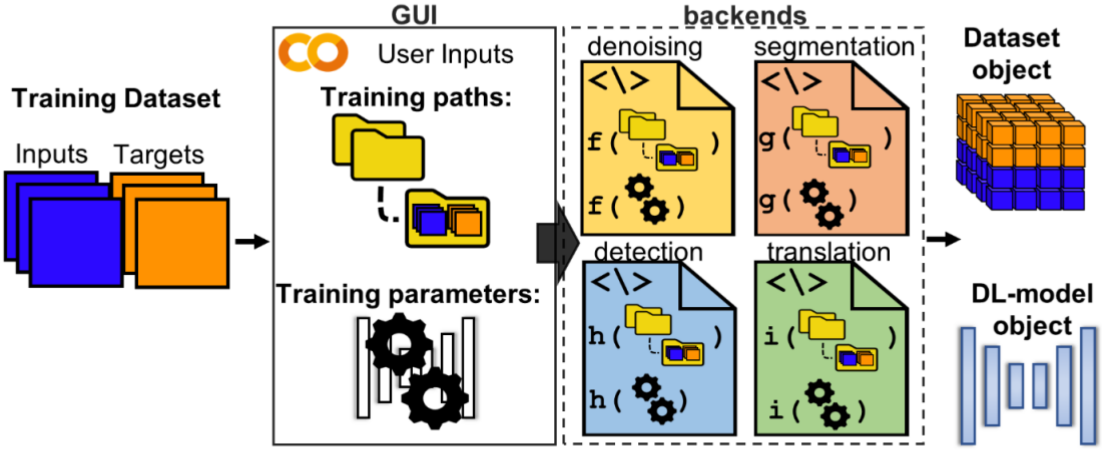
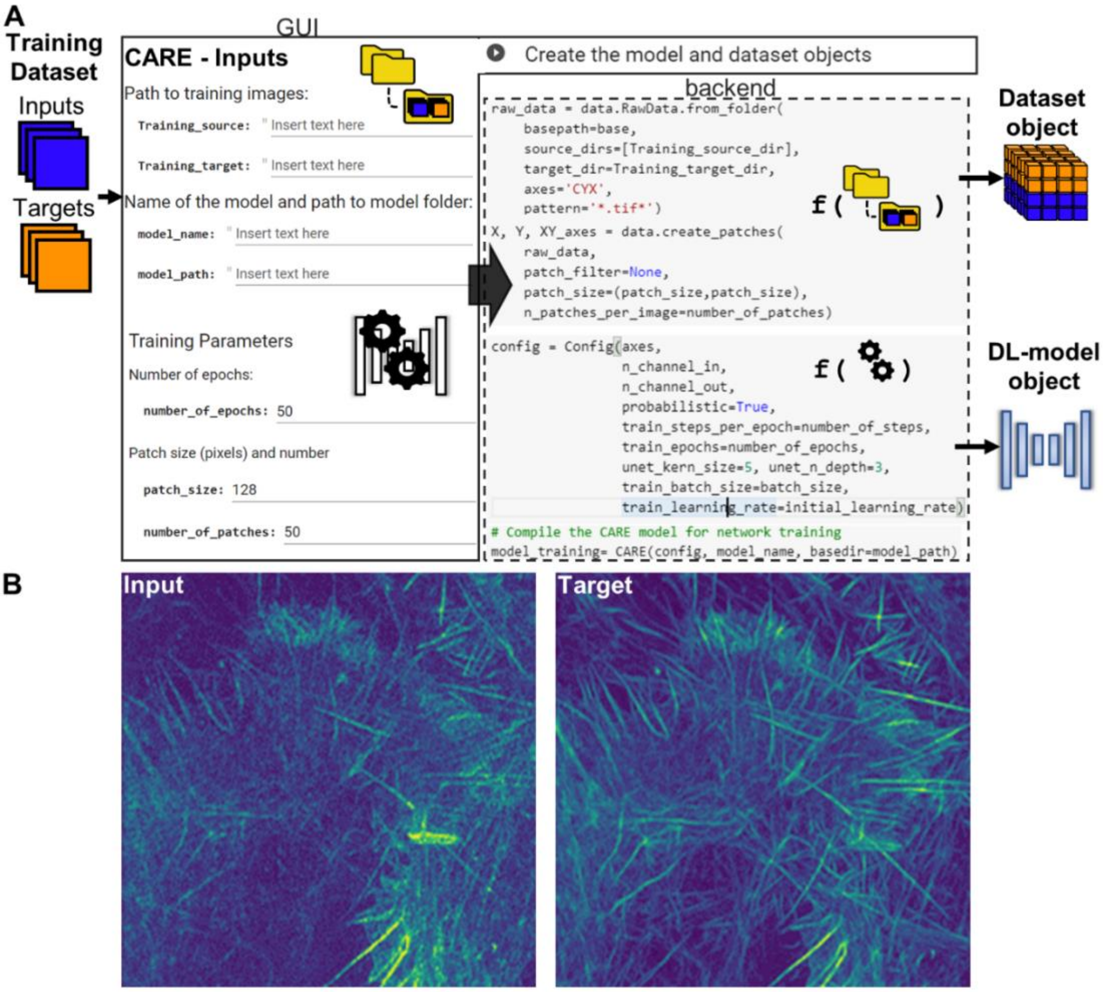


Figure 11 - Envisioned flow of steps 3-4 *- Converting user datasets to DL-objects: The user provides a curated dataset with paired images or images and annotations. The GUI will have a consistent format where training paths for the datasets and model hyperparameters are entered. This is the only inputs the user needs to give. The backends in the different notebooks use different functions to convert these instructions from the GUI into dataset and model objects, ready for training.*

i. CARE 2D

A convenient way to create a dataset object compatible with a DL method is to use tools created by the authors of the method themselves, which convert files, given as a path variable, into the required format. Such functions exist for several methods used in this project, specifically those of the CSBDeep ‘family’ (CARE, N2V, StarDist). As these methods can take a path variable as input and give NumPy arrays that are compatible with the respective neural networks as output, the implementation in a Colab notebook was comparatively simple. The paths to the training dataset are inserted by the user in the ‘Training_source’ and Training_target’ (as shown in Fig. 12) fields and are used as input arguments in these custom functions to convert them into a CARE compatible format. The neural network model object is also created using a CARE-specific function which is imported from CARE’s Python package. The training parameters are defined via a configuration library (Python dict) which stores key-value pairs for each hyperparameter. To implement this into the GUI, the parameters defined in the input fields by the user are used as the values for this configuration library, called c*onfig* (similarly for StarDist 2D: *Config2D*, and N2V: *N2VConfig*).



CARE - Inputs

Path to training images:

Training_source: " Insert text here

Training_target: " Insert text here

Name of the model and path to model folder:

model_name: " Insert text here

model_path: " Insert text here

Training Parameters

Number of epochs:

number_of_epochs: 50

Patch size (pixels) and number

patch_size: 128

number_of_patches: 50

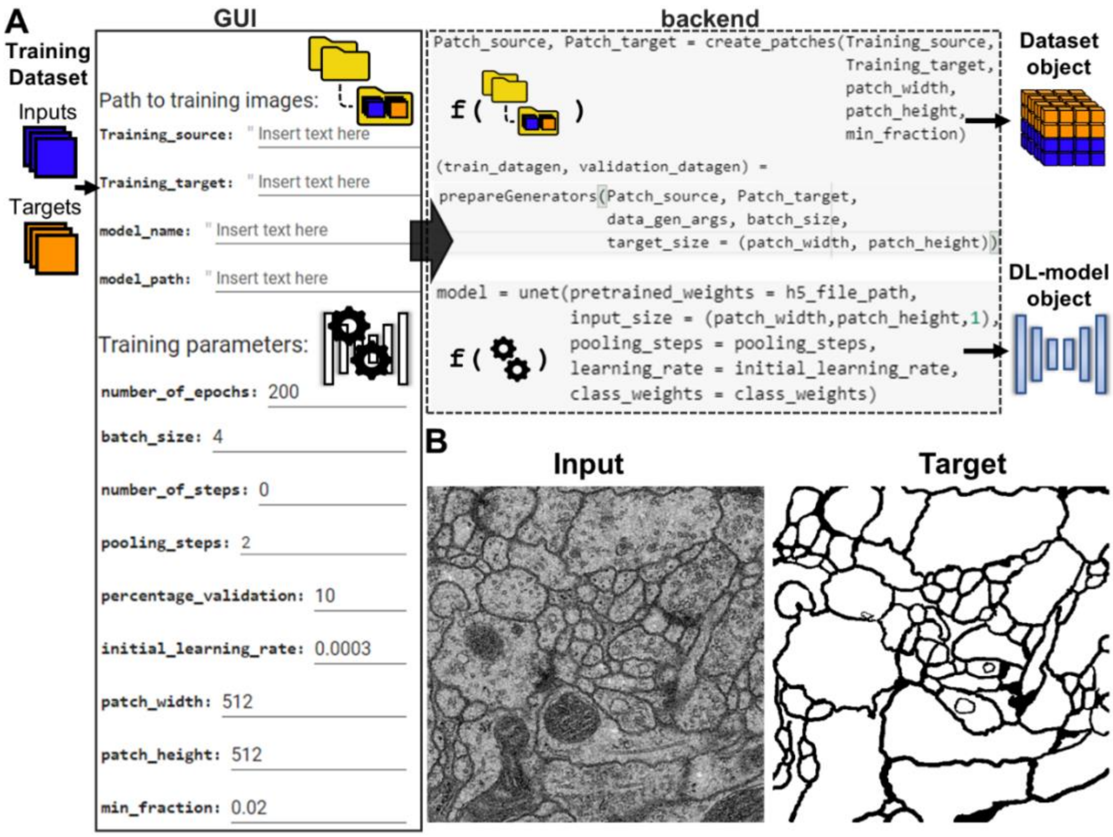
Create the model and dataset objects

```
raw_data = data.RawData.from_folder(
    basepath=base,
    source_dirs=[Training_source_dir],
    target_dir=Training_target_dir,
    axes='CYX',
    pattern='*.tif*')
X, Y, XY_axes = data.create_patches(
    raw_data,
    patch_filter=None,
    patch_size=(patch_size,patch_size),
    n_patches_per_image=number_of_patches)

config = Config(axes,
                n_channel_in,
                n_channel_out,
                probabilistic=True,
                train_steps_per_epoch=number_of_steps,
                train_epochs=number_of_epochs,
                unet_kern_size=5, unet_n_depth=3,
                train_batch_size=batch_size,
                train_learning_rate=initial_learning_rate)
# Compile the CARE model for network training
model_training= CARE(config, model_name, basedir=model_path)
```

Figure 12 - Step 4 - Converting User inputs to DL-objects in CARE 2D notebook* – A)** *CARE functions imported as shown in fig.3 can be used to convert user inputs from folders into the required dataset objects. Additional parameters, here patch size and the number of patches is used in another function to convert the 'rawdata' into patches automatically. Training parameters, e.g. 'number_of_epochs' is used directly as input to create a config object which is used later to create the model object for training (shown on bottom right). This command is executed in the notebook in the Create the model and dataset objects cell (shown on top right)) which is played directly after the training inputs. (For simplicity not all parameters available in the notebook are shown).* ***B) *Output shown after playing the parameter cell in CARE 2D notebook.*

ii. U-Net 2D

To convert paths and training hyperparameters into compatible object in the U-Net notebook was possible by creating Python functions that convert the user's paths and chosen validation splits into Keras data generator objects. Here, Romain Laine and I created custom functions that first split the dataset into patches and then used these patches as NumPy arrays as inputs for a data generator function built around the standard Keras datagenerator object (Fig. 13). An advantage of using the Keras data generator is that it creates training batches of data 'on the fly', i.e. during training, in contrast to making all the patches at once and then passing them to the network. This minimises the use of the available RAM capacity in Colab, which is helpful, as RAM was identified as one of the limitations in Colab[134].



Path to training images:

Training_source: Insert text here

Training_target: Insert text here

model_name: Insert text here

model_path: Insert text here

Training parameters:

number_of_epochs: 200

batch_size: 4

number_of_steps: 0

pooling_steps: 2

percentage_validation: 10

initial_learning_rate: 0.0003

patch_width: 512

patch_height: 512

min_fraction: 0.02

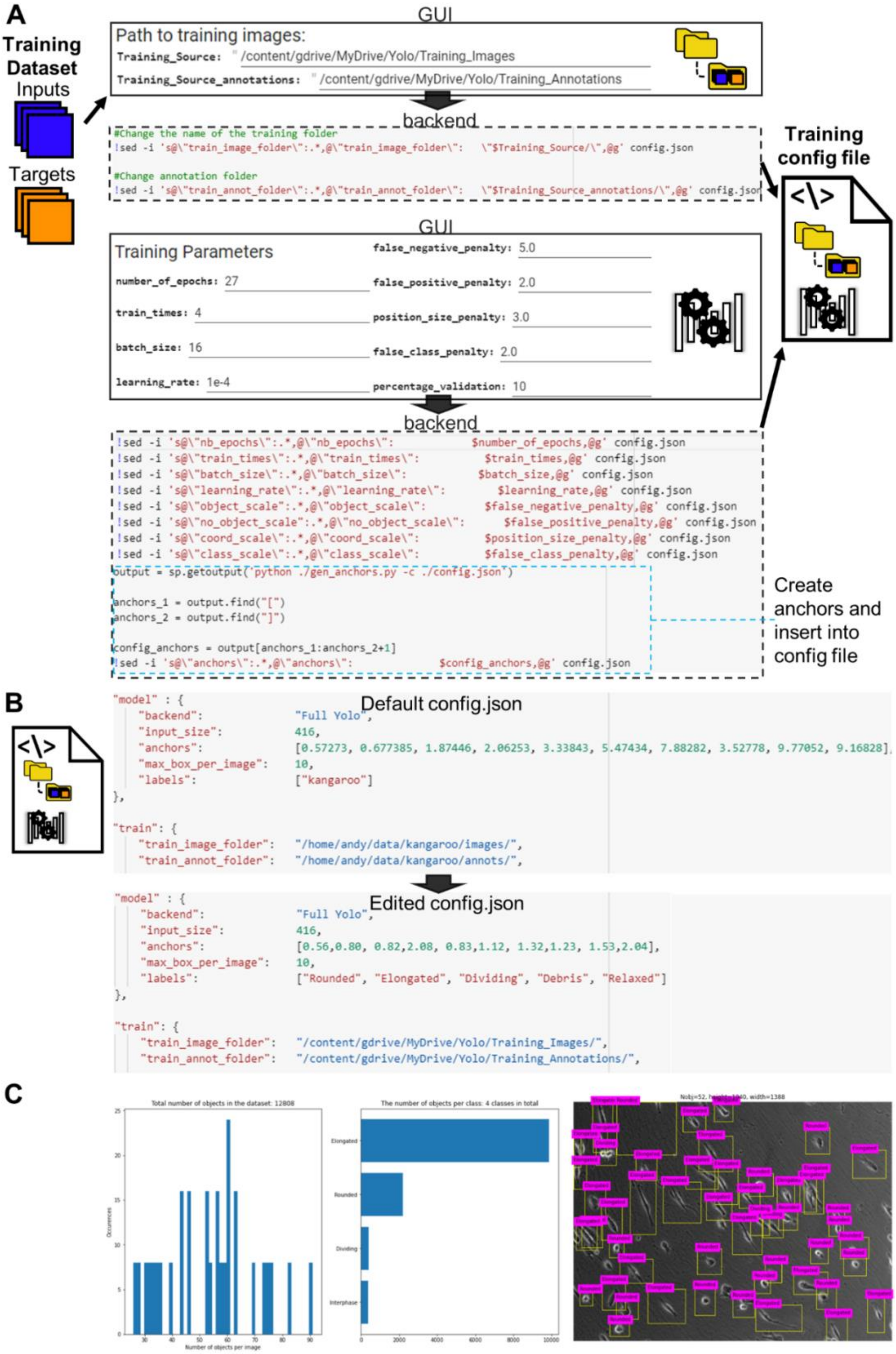
```
Patch_source, Patch_target = create_patches(Training_source,
                                            Training_target,
                                            patch_width,
                                            patch_height,
                                            min_fraction)

(train_datagen, validation_datagen) =
 prepareGenerators(Patch_source, Patch_target,
                   data_gen_args, batch_size,
                   target_size = (patch_width, patch_height))

model = unet(pretrained_weights = h5_file_path,
             input_size = (patch_width,patch_height,1),
             pooling_steps = pooling_steps,
             learning_rate = initial_learning_rate,
             class_weights = class_weights)
```

Figure 13- Step 4 - Converting User inputs to DL-objects in U-Net 2D notebook – ***A)*** *Custom Python functions defined in the installation section for U-Net (shown in fig. 3) are used to create the necessary data objects and the model, using the user's parameters directly.* ***B)*** *Output shown after playing the parameter cell in U-Net 2D notebook.*

iii. YOLOv2

The implementation of methods that define objects via script files is more challenging within the ZeroCostDL4Mic GUI, as it is difficult to edit a file without explicit user interaction.

In YOLOv2, a data generator function takes its arguments from a *config.json* file and then converts the path variables into dataset objects like those created in the U-Net notebook. Creating this file from scratch in the notebook would not have been trivial, given the aim of a code-free GUI. Hence, the default *config.json* file must be edited by inserting the users' input variables when the cell is played. To do this, I used shell search functions (*sed -i*) to locate the path variable field in the *config.json* and replaced it with the user input (Fig. 14A). This approach was chosen because it does not replace parameters which already exist in the file. This protects the file from being edited again if the user plays the cell multiple times while using the same or similar variable names.



Path to training images:

Training_Source: /content/gdrive/MyDrive/Yolo/Training_Images

Training_Source_annotations: /content/gdrive/MyDrive/Yolo/Training_Annotations

```
#Change the name of the training folder
!sed -i 's@\"train_image_folder\":.*,@\"train_image_folder\":   \"$Training_Source/\",@g' config.json

#Change annotation folder
!sed -i 's@\"train_annot_folder\":.*,@\"train_annot_folder\":   \"$Training_Source_annotations/\",@g' config.json
```

Training Parameters

number_of_epochs: 27

train_times: 4

batch_size: 16

learning_rate: 1e-4

false_negative_penalty: 5.0

false_positive_penalty: 2.0

position_size_penalty: 3.0

false_class_penalty: 2.0

percentage_validation: 10

```
!sed -i 's@\"nb_epochs\":.*,@\"nb_epochs\":                 $number_of_epochs,@g' config.json
!sed -i 's@\"train_times\":.*,@\"train_times\":               $train_times,@g' config.json
!sed -i 's@\"batch_size\":.*,@\"batch_size\":                 $batch_size,@g' config.json
!sed -i 's@\"learning_rate\":.*,@\"learning_rate\":             $learning_rate,@g' config.json
!sed -i 's@\"object_scale":.*,@\"object_scale\":              $false_negative_penalty,@g' config.json
!sed -i 's@\"no_object_scale":.*,@\"no_object_scale\":             $false_positive_penalty,@g' config.json
!sed -i 's@\"coord_scale\":.*,@\"coord_scale\":               $position_size_penalty,@g' config.json
!sed -i 's@\"class_scale\":.*,@\"class_scale\":               $false_class_penalty,@g' config.json
output = sp.getoutput('python ./gen_anchors.py -c ./config.json')

anchors_1 = output.find("[")
anchors_2 = output.find("]")

config_anchors = output[anchors_1:anchors_2+1]
!sed -i 's@\"anchors\":.*,@\"anchors\":                  $config_anchors,@g' config.json
```

```
"model" : {
    "backend":              "Full Yolo",
    "input_size":           416,
    "anchors":              [0.57273, 0.677385, 1.87446, 2.06253, 3.33843, 5.47434, 7.88282, 3.52778, 9.77052, 9.16828],
    "max_box_per_image":    10,
    "labels":               ["kangaroo"]
},

"train": {
    "train_image_folder":   "/home/andy/data/kangaroo/images/",
    "train_annot_folder":   "/home/andy/data/kangaroo/annots/",
```

```
"model" : {
    "backend":              "Full Yolo",
    "input_size":           416,
    "anchors":              [0.56,0.80, 0.82,2.08, 0.83,1.12, 1.32,1.23, 1.53,2.04],
    "max_box_per_image":    10,
    "labels":               ["Rounded", "Elongated", "Dividing", "Debris", "Relaxed"]
},

"train": {
    "train_image_folder":   "/content/gdrive/MyDrive/Yolo/Training_Images/",
    "train_annot_folder":   "/content/gdrive/MyDrive/Yolo/Training_Annotations/",
```

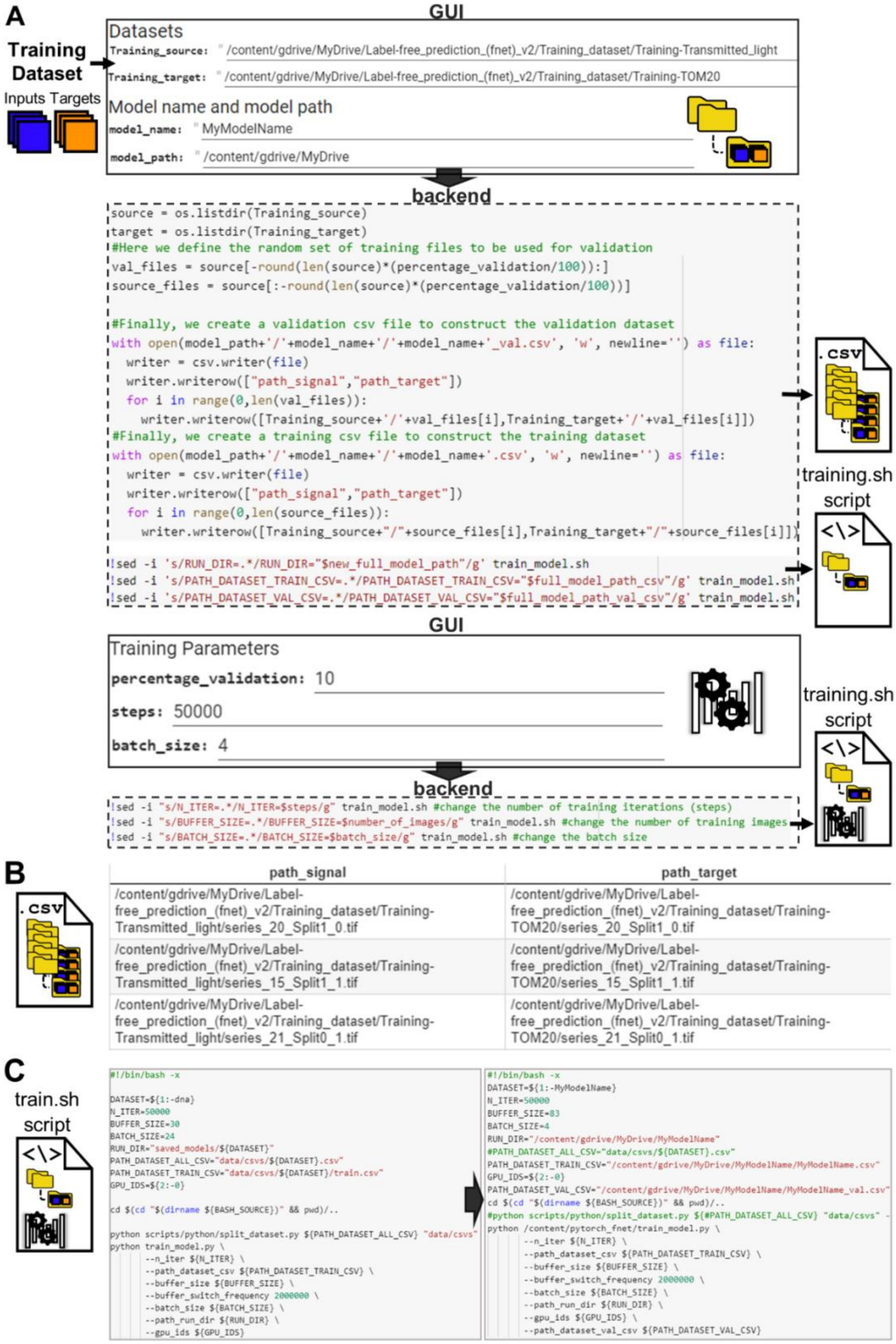
Figure 14 - Step 4 - Converting User inputs to DL-objects in YOLOv2 notebook -* A)** *The paths (top) and hyperparameters (e.g. penalties unique to the YOLOv2 notebook) are edited via shell search command into a 'config.json' which will later be used to create the model (see fig. 10) and dataset objects. YOLOv2 requires the user to calculate anchors for their datasets (blue box) via shell command and copy pasting the result into the 'config.json'. To automate this without user input, the shell output is saved as a* P*ython string ('output') and anchors are extracted from the string through the coordinates of the brackets around them.* ***B) *The main features of the 'config.json' before and after playing the above cell, showing the example paths and new anchors, and class names inserted into the file.* ***C)*** *Output shown after playing the parameter cell in YOLOv2 notebook. Left: Number of objects per image, Middle: Frequency of classes in dataset, Right: Training image with bounding box and class annotations.*

iv. fnet

A similar approach as for YOLOv2 was used to implement label-free prediction or fnet. In fnet, the training function requires at least two files, a configuration file (as YOLOv2), named train.sh, and a train_csv file that contains the path to each training image pair, separated into columns named train_signal and train_target, respectively.

Additionally, the fnet method by default does not include a validation split, and the option to use validation is instead designed to be manually inserted into the train.sh file. Additionally, using validation in this method requires an additional .csv file (*val_csv*) that contains the validation data paths. Creating or editing these files without user interaction required several additional functions to be written to make these methods useable (Fig. 15). As above, I used shell commands to change the train's parameters (as shown above).sh configuration file. However, I wrote additional Python functions that could replace or insert full lines into existing script files as several files in the fnet method needed to be updated. Debugging was simpler when using Python functions to replace lines in Python or other script files. Since it must be assumed that users play cells multiple times when choosing parameters, e.g. when deciding to rename a model, it also needed to be ensured that these functions did not repeatedly insert parameters into the files, as this might have made them unreadable by the functions required to build the dataset and model objects. After thoroughly testing that the replacement and editing functions performed the desired insertions, without throwing errors, they were repeatedly used to change parameters and paths in the script files. Since these functions operate behind the GUI, the user interface can remain consistent with other notebooks.

Unlike U-Net or the CSBDeep methods, the training dataset and model objects are not created after the user chooses the parameters. Because fnet is specifically designed to pass the configuration script to its training function rather than Python objects, the model and dataset objects are created only once the training cell is played. To maintain consistency between notebooks, the cell which constructs the dataset and model objects in U-Net and CARE (*Create model and dataset objects)* is used in the fnet notebooks to create the dataset .csv files with the paths to the dataset. The cell is therefore called *(Create the dataset files for training)*



Datasets

Training_source: " /content/gdrive/MyDrive/Label-free_prediction_(fnet)_v2/Training_dataset/Training-Transmitted_light

Training_target: " /content/gdrive/MyDrive/Label-free_prediction_(fnet)_v2/Training_dataset/Training-TOM20

Model name and model path

model_name: " MyModelName

model_path: " /content/gdrive/MyDrive

```
source = os.listdir(Training_source)
target = os.listdir(Training_target)
#Here we define the random set of training files to be used for validation
val_files = source[-round(len(source)*(percentage_validation/100)):]
source_files = source[:-round(len(source)*(percentage_validation/100))]

#Finally, we create a validation csv file to construct the validation dataset
with open(model_path+'/'+model_name+'/'+model_name+'_val.csv', 'w', newline='') as file:
  writer = csv.writer(file)
  writer.writerow(["path_signal","path_target"])
  for i in range(0,len(val_files)):
    writer.writerow([Training_source+'/'+val_files[i],Training_target+'/'+val_files[i]])
#Finally, we create a training csv file to construct the training dataset
with open(model_path+'/'+model_name+'/'+model_name+'.csv', 'w', newline='') as file:
  writer = csv.writer(file)
  writer.writerow(["path_signal","path_target"])
  for i in range(0,len(source_files)):
    writer.writerow([Training_source+"/"+source_files[i],Training_target+"/"+source_files[i]])

!sed -i 's/RUN_DIR=.*/RUN_DIR="$new_full_model_path"/g' train_model.sh
!sed -i 's/PATH_DATASET_TRAIN_CSV=.*/PATH_DATASET_TRAIN_CSV="$full_model_path_csv"/g' train_model.sh
!sed -i 's/PATH_DATASET_VAL_CSV=.*/PATH_DATASET_VAL_CSV="$full_model_path_val_csv"/g' train_model.sh
```

Training Parameters

percentage_validation: 10

steps: 50000

batch_size: 4

```
!sed -i "s/N_ITER=.*/N_ITER=$steps/g" train_model.sh #change the number of training iterations (steps)
!sed -i "s/BUFFER_SIZE=.*/BUFFER_SIZE=$number_of_images/g" train_model.sh #change the number of training images
!sed -i "s/BATCH_SIZE=.*/BATCH_SIZE=$batch_size/g" train_model.sh #change the batch size
```

path_signal	path_target
/content/gdrive/MyDrive/Label-free_prediction_(fnet)_v2/Training_dataset/Training-Transmitted_light/series_20_Split1_0.tif	/content/gdrive/MyDrive/Label-free_prediction_(fnet)_v2/Training_dataset/Training-TOM20/series_20_Split1_0.tif
/content/gdrive/MyDrive/Label-free_prediction_(fnet)_v2/Training_dataset/Training-Transmitted_light/series_15_Split1_1.tif	/content/gdrive/MyDrive/Label-free_prediction_(fnet)_v2/Training_dataset/Training-TOM20/series_15_Split1_1.tif
/content/gdrive/MyDrive/Label-free_prediction_(fnet)_v2/Training_dataset/Training-Transmitted_light/series_21_Split0_1.tif	/content/gdrive/MyDrive/Label-free_prediction_(fnet)_v2/Training_dataset/Training-TOM20/series_21_Split0_1.tif

```
#!/bin/bash -x

DATASET=${1:-dna}
N_ITER=50000
BUFFER_SIZE=30
BATCH_SIZE=24
RUN_DIR="saved_models/${DATASET}"
PATH_DATASET_ALL_CSV="data/csvs/${DATASET}.csv"
PATH_DATASET_TRAIN_CSV="data/csvs/${DATASET}/train.csv"
GPU_IDS=${2:-0}

cd $(cd "$(dirname ${BASH_SOURCE})" && pwd)/..

python scripts/python/split_dataset.py ${PATH_DATASET_ALL_CSV} "data/csvs"
python train_model.py \
       --n_iter ${N_ITER} \
       --path_dataset_csv ${PATH_DATASET_TRAIN_CSV} \
       --buffer_size ${BUFFER_SIZE} \
       --buffer_switch_frequency 2000000 \
       --batch_size ${BATCH_SIZE} \
       --path_run_dir ${RUN_DIR} \
       --gpu_ids ${GPU_IDS}
```

```
#!/bin/bash -x
DATASET=${1:-MyModelName}
N_ITER=50000
BUFFER_SIZE=83
BATCH_SIZE=4
RUN_DIR="/content/gdrive/MyDrive/MyModelName"
#PATH_DATASET_ALL_CSV="data/csvs/${DATASET}.csv"
PATH_DATASET_TRAIN_CSV="/content/gdrive/MyDrive/MyModelName/MyModelName.csv"
GPU_IDS=${2:-0}
PATH_DATASET_VAL_CSV="/content/gdrive/MyDrive/MyModelName/MyModelName_val.csv"
cd $(cd "$(dirname ${BASH_SOURCE})" && pwd)/..
#python scripts/python/split_dataset.py ${#PATH_DATASET_ALL_CSV} "data/csvs" -
python /content/pytorch_fnet/train_model.py \
       --n_iter ${N_ITER} \
       --path_dataset_csv ${PATH_DATASET_TRAIN_CSV} \
       --buffer_size ${BUFFER_SIZE} \
       --buffer_switch_frequency 2000000 \
       --batch_size ${BATCH_SIZE} \
       --path_run_dir ${RUN_DIR} \
       --gpu_ids ${GPU_IDS} \
       --path_dataset_val_csv ${PATH_DATASET_VAL_CSV}
```

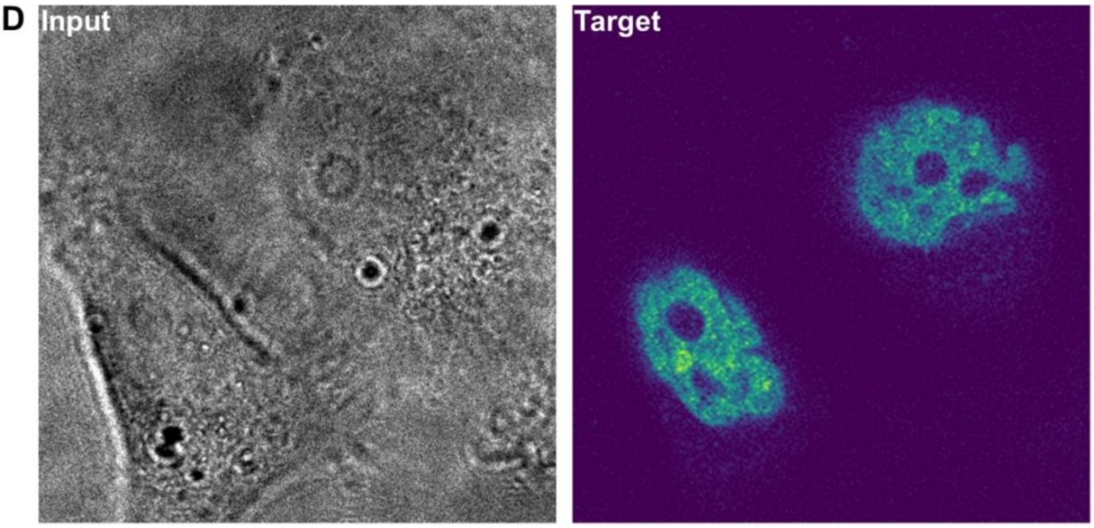
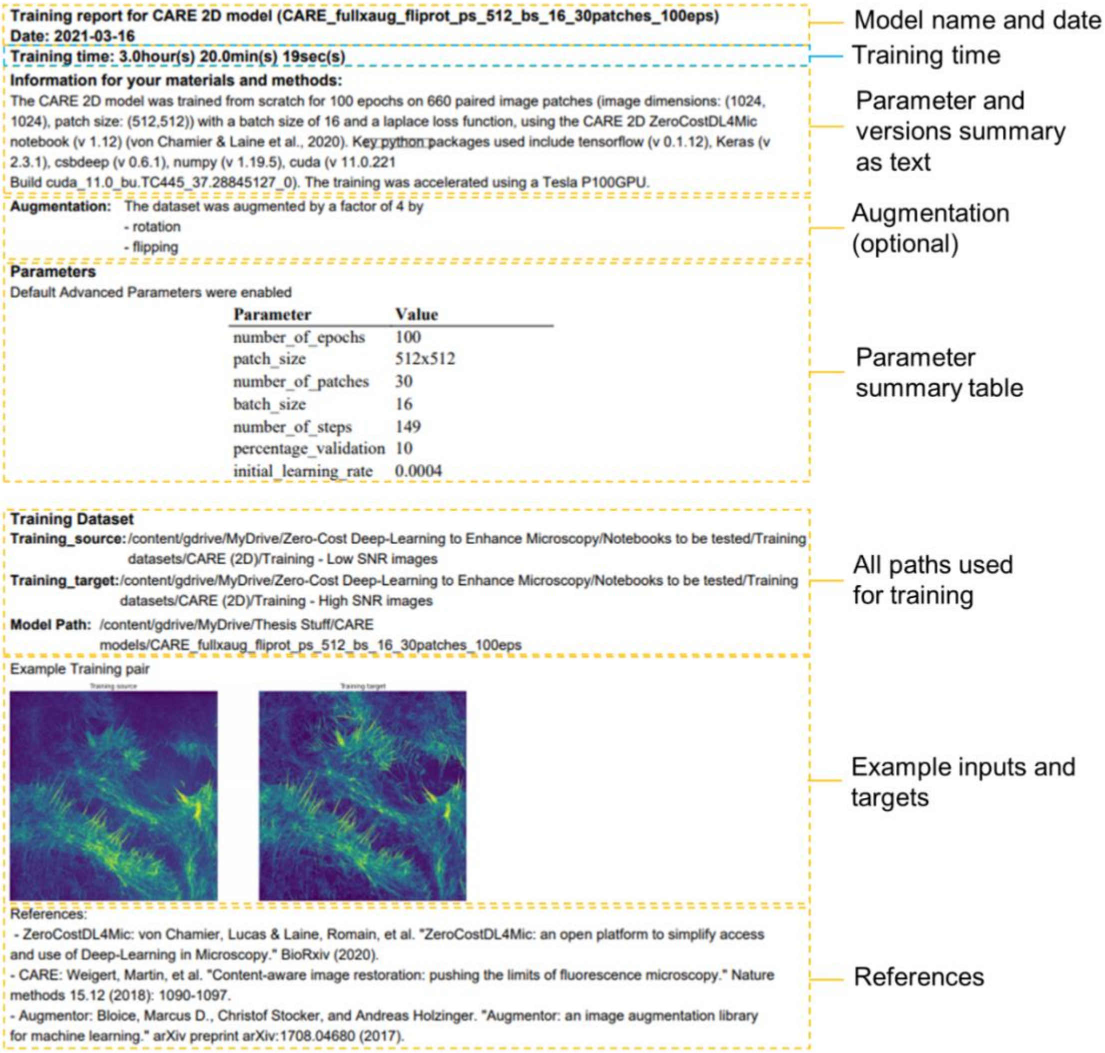


Figure 15 - Step 4 - Converting User inputs to DL-objects in fnet notebook - A) *The user enters the paths to datasets as in other methods. In the backend, the paths are used to create two csv files, one for training and one for validation which carry the model's name. The files used for training and validation are split automatically from the file folders, using the user's choice for validation split (lower GUI). The paths to the csv files, not the actual file folders, are then inserted into a 'train.sh' script file.* ***B)*** *The path .csv file for the training dataset, created in A.* ***C)*** *The 'train.sh' script before (left) and after editing (right) in A.* ***D)*** *Output shown after playing the parameter cell in fnet notebook.*

v. Saving a dataset and hyperparameter summary as pdf file

After choosing training datasets and a set of parameters and training a model, I noticed that it is often difficult to determine precisely which parameters were used to train a model. However, for experimentation with models across days or weeks, this is crucial and having access to these parameters can be essential to compare models' performances and make improvements. Furthermore, to ensure reproducibility of any DL experiments, it is often necessary to give version numbers of the software and the type of hardware used. While some methods that use TensorFlow create an events file that can extract some important features about training (even though this can be cumbersome), details about hardware or versions are usually not included in these files and need to be found by the user. To give all users, regardless of their skills to find parameter values and version numbers the ability to perform analysis across weeks on different models and to report on results quickly, I decided to save this information and details about the datasets and models used in a human-readable pdf file. This output could then also be used easily as a template for any publication of the results, which would simplify reproduction by other users which would be a very attractive feature for the project, as it aims to teach novice users how to train their DL-models. Initially, this file was intended to be saved after training, but the user would only get access if the training cell is played and executes error-free. To accommodate the possibility for such errors or time-outs of the notebook, this pdf is first saved after the parameter choice cell and updated with the time required for training after training completes. This means that after playing the cell in this section of the notebook, the user will have created a pdf file which contains the most important training settings in the folder chosen under the *model_path.* The basis for the pdf creation function (named *pdf_export*) is identical in each notebook, with small adjustments necessary in each notebook to accommodate method-specific parameters. Fig. 16 shows the pdf export of the CARE 2D notebook which is saved after training.



Training report for CARE 2D model (CARE_fullxaug_fliprot_ps_512_bs_16_30patches_100eps)
Date: 2021-03-16

Training time: 3.0hour(s) 20.0min(s) 19sec(s)

Information for your materials and methods:

The CARE 2D model was trained from scratch for 100 epochs on 660 paired image patches (image dimensions: (1024, 1024), patch size: (512,512)) with a batch size of 16 and a laplace loss function, using the CARE 2D ZeroCostDL4Mic notebook (v 1.12) (von Chamier & Laine et al., 2020). Key python packages used include tensorflow (v 0.1.12), Keras (v 2.3.1), csbdeep (v 0.6.1), numpy (v 1.19.5), cuda (v 11.0.221
Build cuda_11.0_bu.TC445_37.28845127_0). The training was accelerated using a Tesla P100GPU.

Augmentation: The dataset was augmented by a factor of 4 by
- rotation
- flipping

Parameters

Default Advanced Parameters were enabled

Parameter	Value
number_of_epochs	100
patch_size	512x512
number_of_patches	30
batch_size	16
number_of_steps	149
percentage_validation	10
initial_learning_rate	0.0004

Training Dataset

Training_source: /content/gdrive/MyDrive/Zero-Cost Deep-Learning to Enhance Microscopy/Notebooks to be tested/Training datasets/CARE (2D)/Training - Low SNR images

Training_target: /content/gdrive/MyDrive/Zero-Cost Deep-Learning to Enhance Microscopy/Notebooks to be tested/Training datasets/CARE (2D)/Training - High SNR images

Model Path: /content/gdrive/MyDrive/Thesis Stuff/CARE models/CARE_fullxaug_fliprot_ps_512_bs_16_30patches_100eps

Example Training pair

References:
- ZeroCostDL4Mic: von Chamier, Lucas & Laine, Romain, et al. "ZeroCostDL4Mic: an open platform to simplify access and use of Deep-Learning in Microscopy." BioRxiv (2020).
- CARE: Weigert, Martin, et al. "Content-aware image restoration: pushing the limits of fluorescence microscopy." Nature methods 15.12 (2018): 1090-1097.
- Augmentor: Bloice, Marcus D., Christof Stocker, and Andreas Holzinger. "Augmentor: an image augmentation library for machine learning." arXiv preprint arXiv:1708.04680 (2017).

Model name and date

Training time

Parameter and versions summary as text

Augmentation (optional)

Parameter summary table

All paths used for training

Example inputs and targets

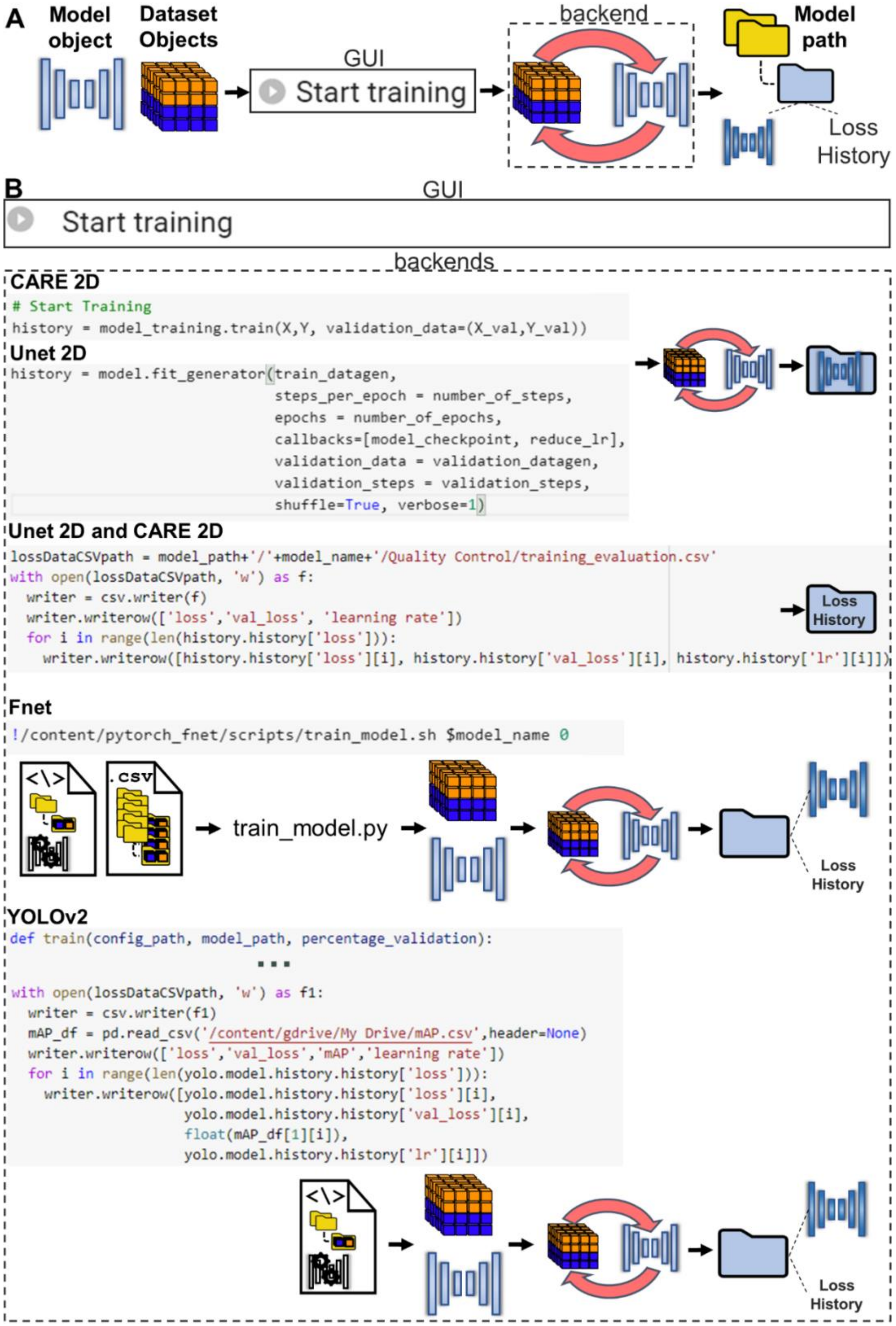
References

Figure 16 - Creation of a pdf summary of training and parameters - *The document exported for the user after choosing parameters and playing the cell (here, for the example of CARE 2D). These sections are conserved across all notebooks. The version of the pdf export shown, with the training time displayed becomes available only after training completes. Similarly, the augmentation option is only displayed if this option is chosen in the notebook.*

II. 2. 1. e. Execution of Training

Aims: In ZeroCostDL4Mic, the training cell must train the respective neural networks without user input, but we also need to create a set of outputs that can be used to evaluate the model after training (Fig. 17A). Specifically, we wanted to save a .csv file after training which would contain the losses of the model per epoch and the learning rate used per epoch. Viewing losses after training is a standard step in performance evaluation in DL tasks (and is also done in ZeroCostdL4Mic's QC step) and TensorFlow even provides this option in the *tensorboard* GUI. However, *tensorboard* is only viewable within the notebook and it is not easy for users to quickly extract the loss information from *tensorboard*. Exporting simple .csv files was therefore thought to facilitate the ability of users to analyse their models after training as these are human-readable and can quickly be used for quantitative studies.

Finally, the training section needs to create the trained model object in the location indicated by the user in the previous section.



```
# Start Training
history = model_training.train(X,Y, validation_data=(X_val,Y_val))
```

```
history = model.fit_generator(train_datagen,
                              steps_per_epoch = number_of_steps,
                              epochs = number_of_epochs,
                              callbacks=[model_checkpoint, reduce_lr],
                              validation_data = validation_datagen,
                              validation_steps = validation_steps,
                              shuffle=True, verbose=1)
```

```
lossDataCSVpath = model_path+'/'+model_name+'/Quality Control/training_evaluation.csv'
with open(lossDataCSVpath, 'w') as f:
  writer = csv.writer(f)
  writer.writerow(['loss','val_loss', 'learning rate'])
  for i in range(len(history.history['loss'])):
    writer.writerow([history.history['loss'][i], history.history['val_loss'][i], history.history['lr'][i]])
```

```
!/content/pytorch_fnet/scripts/train_model.sh $model_name 0
```

```
def train(config_path, model_path, percentage_validation):
                          ...
with open(lossDataCSVpath, 'w') as f1:
  writer = csv.writer(f1)
  mAP_df = pd.read_csv('/content/gdrive/My Drive/mAP.csv',header=None)
  writer.writerow(['loss','val_loss','mAP','learning rate'])
  for i in range(len(yolo.model.history.history['loss'])):
    writer.writerow([yolo.model.history.history['loss'][i],
                     yolo.model.history.history['val_loss'][i],
                     float(mAP_df[1][i]),
                     yolo.model.history.history['lr'][i]])
```

Figure 17 - Step 5 – Execution of Training in ZeroCostDL4Mic - A*) The model objects (or other files) created in previous steps should are to be used for training the model, by playing a single 'Start Training' cell. All outputs from training are saved in the model folder (blue).* ***B)*** *In CARE and U-Net notebooks, the Start Training cell executes training functions which directly create a* Keras *model object from which the loss history is extracted after training. In fnet, 'Start Training' executes a simple script from the fnet repository which automatically saves model checkpoints and losses in the model folder. YOLOv2 executes a training function defined in the notebook, part of which creates the loss .csv file after training is complete, here, also including the mAP metric in addition to losses. (The full training function can be viewed in the YOLOv2 notebook: https://colab.research.google.com/github/HenriquesLab/ZeroCostDL4Mic/blob/master/Colab_notebooks/YOLOv2_ZeroCostDL4Mic.ipynb*

i. CARE 2D

In CARE, the training function can be called from the model objects (*model_training*) and creates a Keras history object that can be used after training completes to extract the losses and learning rates stored in a csv file once per epoch (Fig. 17B top). Hence, after the model object and dataset objects are created in the previous section, training can be initiated by calling the training functions on the model object. Implementation was simple as no user input is required and the user will only need to play the training cell to train the CARE model. During training the best weights, i.e. those weights which reduce the loss on the validation dataset after each epoch, are updated and saved in the folder called *model_name* in the parent folder *model_path*.

ii. U-Net 2D

In U-Net 2D, a standard Keras training function (*fit_generator*) is used to train the model on the data created by the data generator defined in the previous section. Most of the users chosen parameters can be added directly as input arguments of the *fit_generator* function. As in the CARE notebook, losses can be extracted from the history object after training.

iii. YOLOv2

In YOLOv2, several changes to the default training function were needed to align it with the goals outlined for this section. This was necessary because in the YOLO repository two different functions are used for training: one is a high-level Python training function definition that is designed to be run as a shell script. This function calls the underlying second training function which belongs to the YOLO class object that creates the neural network model. This second function is an adaptation of Keras' default training function (as in U-Net 2D). Running shell scripts in the Colab notebook can make it difficult to access the file objects created during training such as the loss histories and thus requires changes which allow such useful files to be stored by for the user. In fnet, which by default also runs training via shell script, changes did not need to be made because its training function automatically stored loss .csv files which was not the case for the YOLOv2's default shell training function. I therefore transferred the code of this high-level training function to the Colab notebook, so it could be run as a standard Python function without shell commands. This allowed me to add code that would create loss csv files, exploiting the same Keras history object as CARE and U-Net, in this case as an element of the trained YOLO class object.

Inserting the training function into the notebook also facilitated further changes useful in ZeroCostDL4Mic. Specifically, I added mean average precision (mAP) as an additional validation metric for training. As a relatively intuitive metric, inspecting mAP per training epoch makes it easier for users to interpret how well the model learns to classify unseen data (validation dataset) than the relatively complex loss evaluated during YOLOv2 training[94]. This would also allow for saving two different models at the end of training, with the minimal validation loss and the one with the best validation mAP. Since neither losses nor mAP on the validation set during training will predict the model's performance on unseen data, using these two metrics gives users more opportunity to choose between model weights that optimise performance. Adding mAP required changes to the underlying training function of the YOLO class which is created in the frontend.py of the YOLO repository. No mAP evaluation function was included in the main branch of the YOLO GitHub repository. Therefore, I used a side-branch that included mAP evaluation as a custom Keras callback and included it into a cloned repository in Colab. This was necessary as the side-branch contained several bugs not present in the main branch and could therefore not be used as the default version in the ZeroCostDL4Mic notebook. I, therefore, cloned the mAP callback script from the side-branch and copied it into the main repository in Colab during installation. To use the mAP callback, all YOLO files needed to be updated, e.g. by adding the mAP callback to the imported functions at the start of each script, specifically YOLO’s frontend.py required for training. These changes all occur during installation without interaction by the user, i.e. occur code-free. Due to these changes, after training, the user of the ZeroCostDL4Mic YOLOv2 notebook has access to all training losses and mAP values as .csv files after training and to models with weights optimising either validation loss or mAP (Fig. 17B bottom).

iv. fnet

In fnet, the execution of training usually requires running a Python script via shell command. This process can be implemented in the Colab environment by adding an exclamation mark to the beginning of the line that executes the Python training script. The Python script (train_model.py) then creates the model and dataset objects by parsing the user's input from the configuration script (train.sh) and the csv files containing the paths to the training and validation datasets, all of which are created or edited during the parameter and training path choices (see above) and the following cells (Fig. 17B, third from top).

In fnet, no history object is created and only the trained model is available after training, with no Python objects stored in the runtime. However, the losses are stored during training by default in fnet in two dynamically updated .csv files (for training and validation) updated dynamically. However, unlike CARE and U-Net, fnet's validation losses are not evaluated per epoch because fnet executes training differently from the other networks used in ZeroCostDL4Mic. Instead of dividing the dataset into a specific set of batches repeatedly used over multiple epochs to train the model, in fnet the dataset is split into a much higher number of unique batches (although redundancy can be expected) used only once. Therefore, the training of fnet is essentially one very long epoch in which eventually all the batches of the data set are 'seen'. Improvement of the model weights is not usually seen per step (i.e. batch), but over long batches. Hence, I set the evaluation of validation batches to occur only once every 100 steps, which means that the validation loss.csv has 100 times fewer entries than the training loss.csv. This is the only key difference in the output from the training cell for users in fnet compared to the other notebooks where all losses are stored in one csv file. Learning rates are not stored for fnet as it is not changed during training, i.e., there is no equivalent to the reduction of the learning rate once the loss reaches a plateau (which is done, e.g. in CARE and U-Net).

II. 2. 1. f. Performing Quality Control

Aims: Quality Control (QC) is a central feature in ZeroCostDL4Mic with several uses within the platform and importance for democratising DL methods (discussed more in chapter III). QC aims to provide users with enough information to judge whether a trained DL-model can be trusted to perform a task on an unseen dataset. In ZeroCostDL4Mic, we want to achieve this in the following way:

- The user chooses a model they want to evaluate in the GUI

- The user curates a test dataset with inputs and targets, separate from the training dataset, and uploads this dataset to the Google Drive. The user can then input the paths to these test input and test target images in the GUI.

- The QC should provide the following results:

    - training history of the model

    - predictions on the provided dataset (QC dataset)

    - quantitative evaluation of differences between model predictions and targets

    - example representation of differences between predictions and targets (qualitative evaluation)

These results should be displayed in the notebook and need to be easily accessible by the user for later use. This envisioned workflow for QC is shown schematically in Fig. 18.

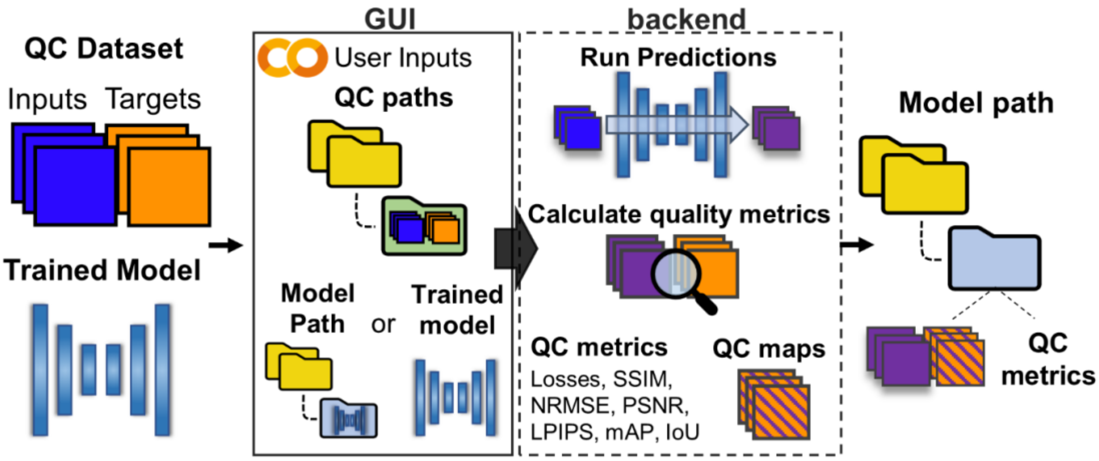
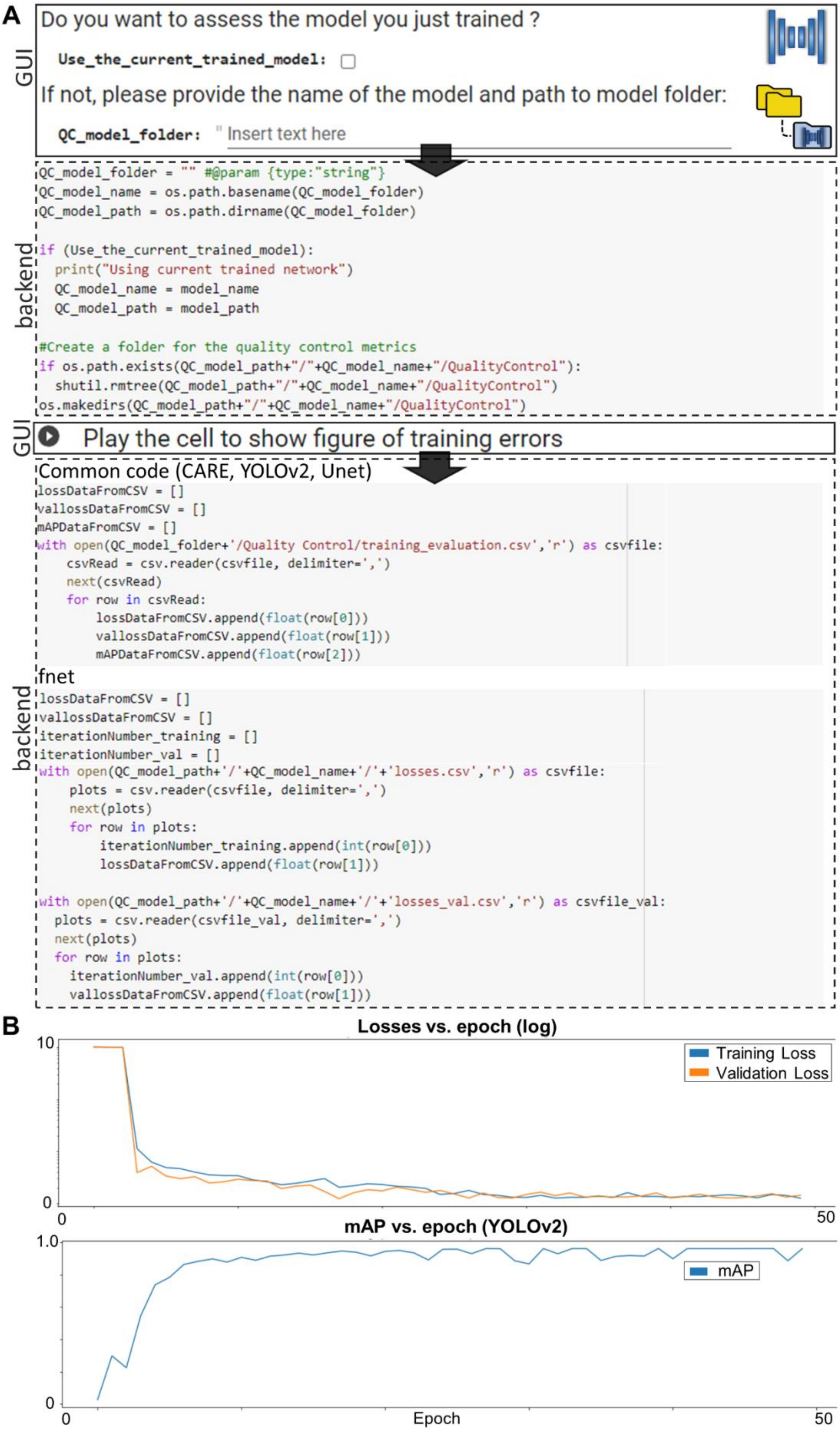


Figure 18 - Envisioned flow of step 6 - Quality Control (QC) step in ZeroCostDL4Mic *- The user curates a QC dataset from the same domain as the training dataset and has access to a trained model. In the GUI, the user provides paths to this dataset and chooses either a trained model from the current runtime or the path to a weights file from a trained model. In the notebook backend, the model is used to create predictions on the inputs, calculates metrics (different metrics for different notebooks), and where applicable also creates and saves error maps. All results are saved inside the model folder.*

The first step of the QC section is implemented almost identically across all notebooks. The user first selects the model to be evaluated. Here, we implemented a choice for the user: either using the most recently trained model in the Colab runtime (using a checkbox) or choosing a model previously saved by the user (by inserting the path to the folder in which the model is stored in). The former option is only available to the user if they have trained a model in the notebook in the same runtime; the latter can also be used if an old model needs to be evaluated without training, providing independence of the section from the rest of the notebook. In this cell, the notebook will create a new QC folder in the '*model_folder*' (if Option 1 is used) or the '*QC_model_folder*' in the latter case and will overwrite any existing files under the same path. This ensures that only the files from the current QC session are stored and avoids any possible confusion with previous QC trials (Fig. 19A top).

Next, the .csv files with the training and validation loss which are stored from the training section (see above) are used to plot the loss history of the model which helps identify potential overfitting during training. Here, we plot both on a linear scale and logarithmic scale, in the latter case to visualise potentially incremental changes in losses (Fig. 19B, log scale not shown). These are consistent across notebooks with the exception for the fnet notebook, which stores fewer validation loss values than training loss values (see above). GAN notebooks that do not store classic losses are evaluated differently.



Do you want to assess the model you just trained ?

Use_the_current_trained_model: □

If not, please provide the name of the model and path to model folder:

QC_model_folder: " Insert text here

```
QC_model_folder = "" #@param {type:"string"}
QC_model_name = os.path.basename(QC_model_folder)
QC_model_path = os.path.dirname(QC_model_folder)

if (Use_the_current_trained_model):
  print("Using current trained network")
  QC_model_name = model_name
  QC_model_path = model_path

#Create a folder for the quality control metrics
if os.path.exists(QC_model_path+"/"+QC_model_name+"/QualityControl"):
  shutil.rmtree(QC_model_path+"/"+QC_model_name+"/QualityControl")
os.makedirs(QC_model_path+"/"+QC_model_name+"/QualityControl")
```

Play the cell to show figure of training errors

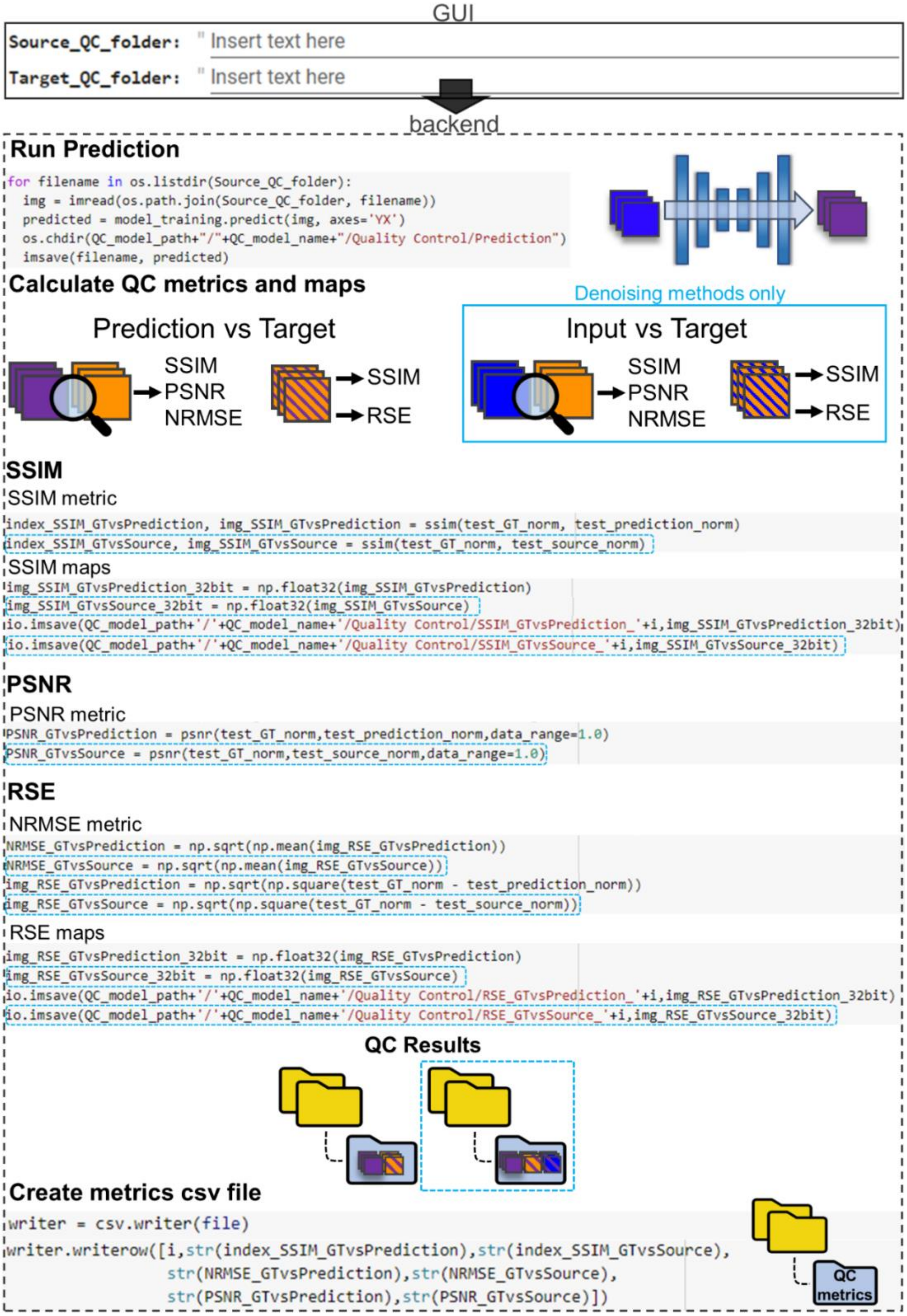
Common code (CARE, YOLOv2, Unet)

```
lossDataFromCSV = []
vallossDataFromCSV = []
mAPDataFromCSV = []
with open(QC_model_folder+'/Quality Control/training_evaluation.csv','r') as csvfile:
    csvRead = csv.reader(csvfile, delimiter=',')
    next(csvRead)
    for row in csvRead:
        lossDataFromCSV.append(float(row[0]))
        vallossDataFromCSV.append(float(row[1]))
        mAPDataFromCSV.append(float(row[2]))
```

fnet

```
lossDataFromCSV = []
vallossDataFromCSV = []
iterationNumber_training = []
iterationNumber_val = []
with open(QC_model_path+'/'+QC_model_name+'/'+'losses.csv','r') as csvfile:
    plots = csv.reader(csvfile, delimiter=',')
    next(plots)
    for row in plots:
        iterationNumber_training.append(int(row[0]))
        lossDataFromCSV.append(float(row[1]))

with open(QC_model_path+'/'+QC_model_name+'/'+'losses_val.csv','r') as csvfile_val:
  plots = csv.reader(csvfile_val, delimiter=',')
  next(plots)
  for row in plots:
    iterationNumber_val.append(int(row[0]))
    vallossDataFromCSV.append(float(row[1]))
```

Figure 19 - Step 6 – part 1 - Extracting and displaying Losses in the QC section – A) *The user has the choice between a model trained in the runtime (tickbox) or a saved model (QC_model_folder). In the backend, the QC_model_folder is split into its parent and daughter folder or the model_name and model_path chosen in the input cell of the runtime (fig.*13-15*) are used. Any existing QC folders are removed. Next, the losses are extracted from the loss .csv files which are stored after training when the user plays the second cell Play the cell to show figure of training errors. This backend is identical across all notebooks, the only exception being fnet, where the losses are extracted from two separate .csv files. The losses are then displayed below the cell as output.* ***B)*** *example losses from a YOLOv2 training. For simplicity the losses are shown only in logarithmic scale, although linear scale is also displayed in all notebooks. mAP is only displayed in the YOLOv2 notebook.*



Run Prediction

```
for filename in os.listdir(Source_QC_folder):
  img = imread(os.path.join(Source_QC_folder, filename))
  predicted = model_training.predict(img, axes='YX')
  os.chdir(QC_model_path+"/"+QC_model_name+"/Quality Control/Prediction")
  imsave(filename, predicted)
```

Calculate QC metrics and maps

SSIM

SSIM metric

```
index_SSIM_GTvsPrediction, img_SSIM_GTvsPrediction = ssim(test_GT_norm, test_prediction_norm)
index_SSIM_GTvsSource, img_SSIM_GTvsSource = ssim(test_GT_norm, test_source_norm)
```

SSIM maps

```
img_SSIM_GTvsPrediction_32bit = np.float32(img_SSIM_GTvsPrediction)
img_SSIM_GTvsSource_32bit = np.float32(img_SSIM_GTvsSource)
io.imsave(QC_model_path+'/'+QC_model_name+'/Quality Control/SSIM_GTvsPrediction_'+i,img_SSIM_GTvsPrediction_32bit)
io.imsave(QC_model_path+'/'+QC_model_name+'/Quality Control/SSIM_GTvsSource_'+i,img_SSIM_GTvsSource_32bit)
```

PSNR

PSNR metric

```
PSNR_GTvsPrediction = psnr(test_GT_norm,test_prediction_norm,data_range=1.0)
PSNR_GTvsSource = psnr(test_GT_norm,test_source_norm,data_range=1.0)
```

RSE

NRMSE metric

```
NRMSE_GTvsPrediction = np.sqrt(np.mean(img_RSE_GTvsPrediction))
NRMSE_GTvsSource = np.sqrt(np.mean(img_RSE_GTvsSource))
img_RSE_GTvsPrediction = np.sqrt(np.square(test_GT_norm - test_prediction_norm))
img_RSE_GTvsSource = np.sqrt(np.square(test_GT_norm - test_source_norm))
```

RSE maps

```
img_RSE_GTvsPrediction_32bit = np.float32(img_RSE_GTvsPrediction)
img_RSE_GTvsSource_32bit = np.float32(img_RSE_GTvsSource)
io.imsave(QC_model_path+'/'+QC_model_name+'/Quality Control/RSE_GTvsPrediction_'+i,img_RSE_GTvsPrediction_32bit)
io.imsave(QC_model_path+'/'+QC_model_name+'/Quality Control/RSE_GTvsSource_'+i,img_RSE_GTvsSource_32bit)
```

Create metrics csv file

```
writer = csv.writer(file)
writer.writerow([i,str(index_SSIM_GTvsPrediction),str(index_SSIM_GTvsSource),
                str(NRMSE_GTvsPrediction),str(NRMSE_GTvsSource),
                str(PSNR_GTvsPrediction),str(PSNR_GTvsSource)])
```

Figure 20 - Step 6 – part 2 – Calculating Quality metrics and error maps in the QC section - *The user enters the paths to the QC inputs and targets in the GUI. In the backend, the model performs predictions on the inputs. Images are then compared using quality control functions SSIM, PSNR and RSE and error maps are calculated (see Fig. 21). All results are saved in the model folder. Average metrics are saved as .csv file in QC folder of the model. Circled in blue are comparisons between source and target which are specific to denoising models. For other image-to-image methods, this comparison is not evaluated because inputs and targets cannot necessarily be compared as noisy and low-noise images can be.*

In the second part of the QC, the user inputs their QC input and target dataset paths used to calculate metrics and display examples for visualisation (Fig. 20 top). The evaluation of loss metrics, e.g. between predictions and targets, depends on the modality of data each method uses. It is therefore sensible to discuss these metrics one method at a time.

i. CARE 2D

In denoising methods for bioimaging, such as CARE, inputs, targets, and predictions all represent images from the same modality, usually a fluorescence microscopy image. To calculate the model's performance, we need to primarily evaluate the differences between predictions and targets, which will show how close the network approaches a ground-truth. However, we can also evaluate the difference between input and target which will show how much closer the prediction is to the target than the noisy input. We chose three widely used image quality metrics to evaluate the differences in the ZeroCostDL4Mic notebook: normalised root-mean-squared error (NRMSE), peak signal-to-noise ratio (PSNR) and structural similarity (SSIM). The first two evaluate pixel-wise differences between the images. NRMSE gives the result as a raw calculation of these differences. PSNR gives this result as the ratio between error and signal, giving a slightly more insightful predictor of image quality. SSIM evaluates the images based on the similarities between structures on the images, which is based on a larger field of view around each pixel in the image. (For details on the calculation of these metrics, see Methods in Chapter 2).

The model first needs to run predictions to calculate these metrics. The next section describes in more detail how the prediction is executed. The predictions are saved in a prediction subfolder within the above-created Quality Control folder.

Once the predictions are created, the quality metrics can be evaluated. All three metrics are calculated for each pair of prediction-target and input-target and are stored in a .csv file in the model's ‘Quality Control’ folder. In addition to these metrics, we also create error maps. Since the images are evaluated pixel-by-pixel (in NRMSE and PSNR, directly and in SSIM, via the surrounding pixels), we can visually display the differences between the images as ‘maps’ (Fig. 21). The reason is that the metrics alone give only an average value of the model's performance on the image, which may disguise nuances, such as better or worse predictions in different image regions.

Error maps should highlight these differences or help detect artefacts or weaknesses which could easily be overlooked when studying only the metrics. The same map can be used for the NRMSE and PSNR metrics as both metrics essentially use the same values for their calculation (see Methods in chapter 2). However, instead of averaging over the image we show the root-squared-error (RSE) between the images pixel-by-pixel. This represents an intuitive way to display the unsigned difference in pixel intensities between the images. For the SSIM maps, we use the *structural_similarity* function of the *scikit-image.metrics* package[192] which is based on the SSIM metric as defined in [193,194]. The function is integrated into a custom SSIM function created for the ZeroCostDL4Mic notebook, which simultaneously calculates the average SSIM and SSIM maps.

All the resulting error maps are saved in the Quality Control folder. An example for the quality control evaluation is shown to the user after the cell finishes execution. Here, the user will see an input-target pair, the model's prediction on the input, and RSE and SSIM error maps between input-target and prediction-target pairs, together with the average metrics displayed for the example image.

Together, these results should provide sufficient quantitative and qualitative information on model performance to decide whether the model is suitable for denoising unseen datasets or if a different model needs to be trained.

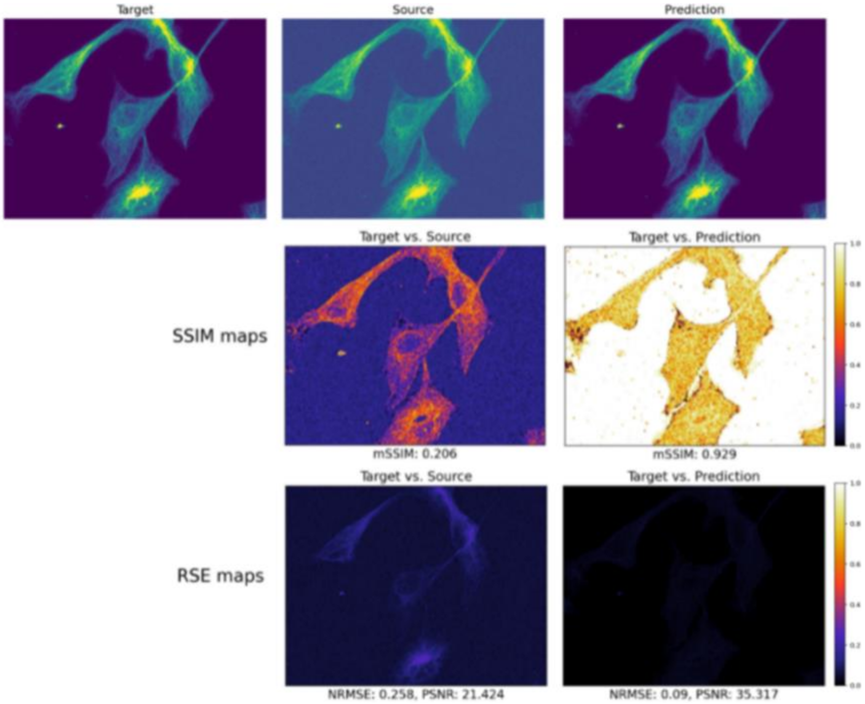


***Figure 21 - Example QC Outputs of CARE 2D notebook** – An equivalent figure is shown to the user after playing section shown in fig. 14 for CARE 2D.*

ii. U-Net 2D

U-Net is designed for semantic segmentation challenges, specifically to distinguish background from structures of interest. One of the most common evaluation metrics for such segmentation challenges is intersection over union (IoU), which measures how many predicted pixels are correctly allocated as foreground/object or background, compared to number of all the predicted pixels (for details, see Methods in chapter 2)[195]. The higher the value the better the prediction.

To evaluate the IoU for U-Net, the output of the non-binary prediction by U-Net models first need to be converted to a binary segmentation map which we do by thresholding. Here, the QC is used to determine the threshold yielding the highest IoU between prediction and target (Fig. 22). This is done by converting the prediction into a binary image in a loop where each iteration increases the signal threshold by one and calculates IoU for each threshold. This is done for each image in the QC dataset, yielding unique threshold curves for each image and an average best threshold for the whole QC dataset. The predicted segmentation map is shown side-by-side with the target segmentation and an overlay between the two, allowing the user to visually investigate the differences between target and prediction (Fig. 22).

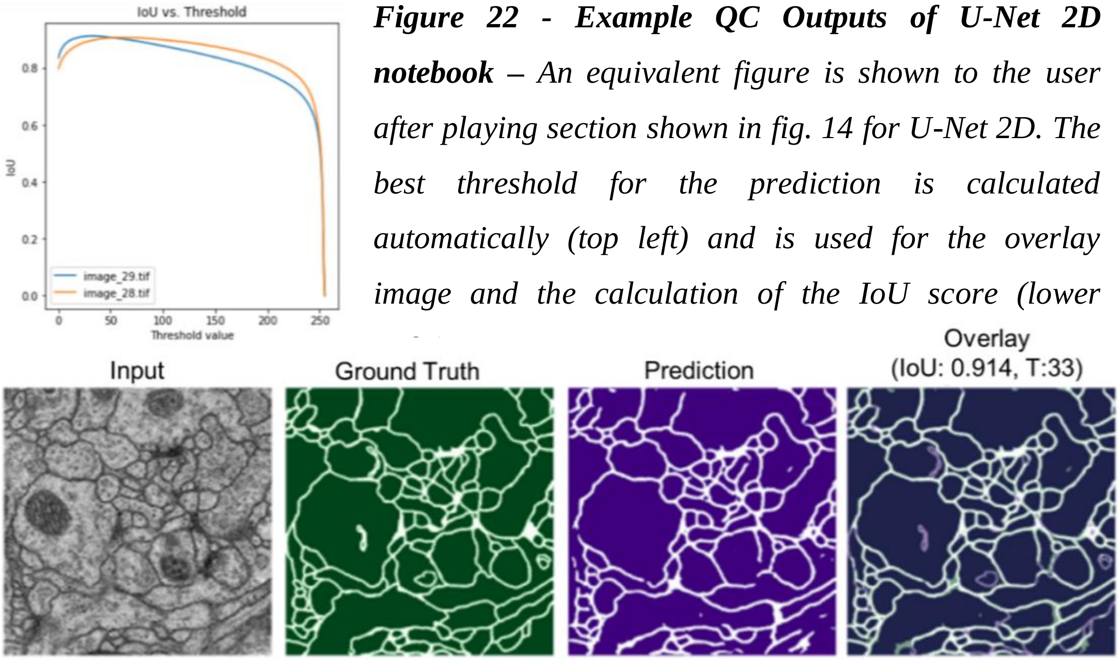


Figure 22 - Example QC Outputs of U-Net 2D notebook *– An equivalent figure is shown to the user after playing section shown in fig. 14 for U-Net 2D. The best threshold for the prediction is calculated automatically (top left) and is used for the overlay image and the calculation of the IoU score (lower*

iii. YOLOv2

In most object detection challenges, the metrics used for performance evaluation are the number of false positives, true positives, and false negatives as these are crucial for real world problems of object detection. Determining these values requires a definition for what counts as a positive detection. In the ZeroCostDL4Mic notebook, we used the IoU between predicted and target bounding boxes with a threshold of 0.4, i.e. at least 40% overlap, and a correct predicted object class to count a detection as a true positive. On the contrary, predicted bounding boxes with a lower IoU to a ground-truth bounding box count as false negatives and predicted bounding boxes with an IoU of 0 count as false positives. From these initial values, additional metrics can be calculated that can give information about the model's performance (Fig. 17A). In many object-detection challenges, the standard metrics which are evaluated are the recall and precision which are calculated as[89]:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

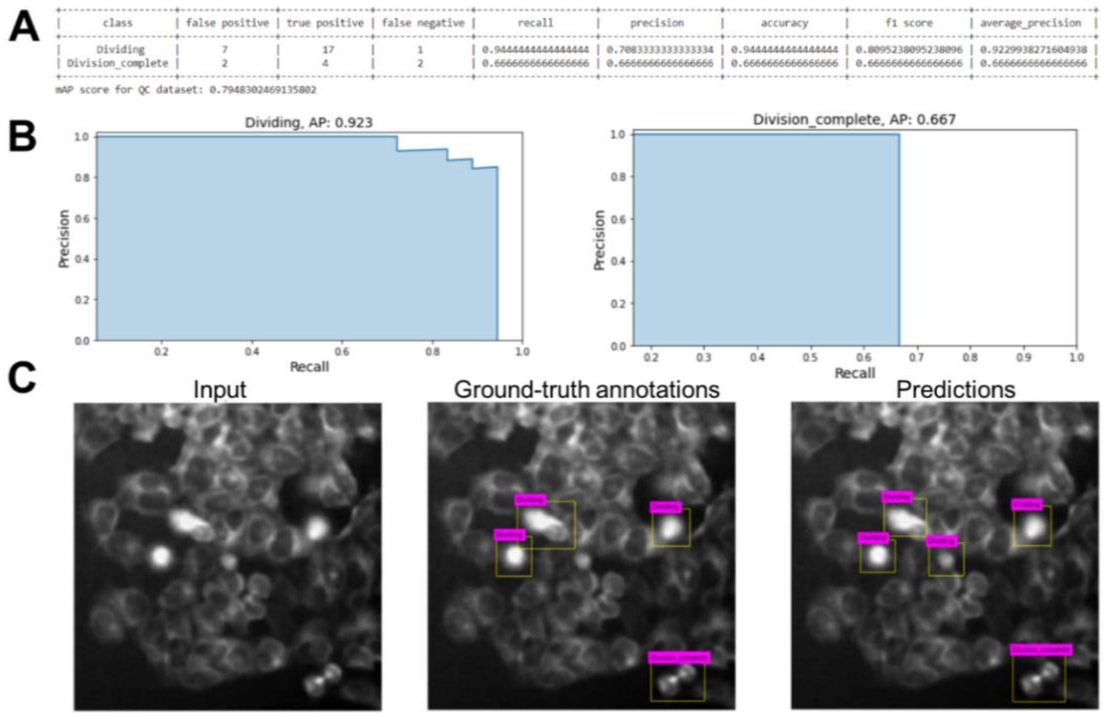
where *TP* – true positive, *FP* – false positive, *FN* – false negative

The so-called precision-recall curve is an additional metric to evaluate the performance (Fig. 17B). To calculate the values on this curve, each object predicted on the QC dataset is ranked by the model's confidence that it predicted it correctly. Next, the recall and precision of the model are calculated one by one. As items with lower confidence scores are evaluated, the precision of the model usually decreases until every predicted object has been evaluated in this way. When recall and precision are plotted against each other in this way, it gives the user an additional quality feature of the network which can be interpreted as how well the model predicts correctly the existence and identity of difficult objects in the dataset which would usually be located lower in the confidence list. High precision for low recall is expected for objects which are relatively easy to classify. In contrast, it is a better indicator of performance if precision remains high as a larger number of objects is taken into account. Aside from the qualitative information drawn from such p-r curves, the quantitative measure that can be extracted from this curve is the average precision of the model on the object class, which is determined by calculating the area under the p-r curve. Once these curves are estimated for all classes to be evaluated in a dataset, the mean value of these gives the mean average precision (mAP) of the model on the dataset which summarises all the other metrics. The other metric that more directly summarises the true and false positives and false negatives is the F1 metric which is calculated as:

$$F1 = 2\,\frac{precision\ x\ recall}{precision + recall}$$

and can give the user additional information to quickly compare the performance of the models with each other.

The evaluated metrics for all objects and each class individually and the p-r curves for all classes are calculated for YOLOv2 and shown as output below the QC cell (Fig. 23A-C) and are stored as .csvs and .pngs, respectively, in the evaluated model's 'Quality Control' folder.



```
+-------------------+----------------+---------------+----------------+--------------------+--------------------+--------------------+--------------------+--------------------+
|       class       | false positive | true positive | false negative |       recall       |     precision      |      accuracy      |      f1 score      | average_precision  |
+-------------------+----------------+---------------+----------------+--------------------+--------------------+--------------------+--------------------+--------------------+
|      Dividing     |       7        |      17       |       1        | 0.9444444444444444 | 0.7083333333333334 | 0.9444444444444444 | 0.8095238095238096 | 0.9229938271604938 |
| Division_complete |       2        |       4       |       2        | 0.6666666666666666 | 0.6666666666666666 | 0.6666666666666666 | 0.6666666666666666 | 0.6666666666666666 |
+-------------------+----------------+---------------+----------------+--------------------+--------------------+--------------------+--------------------+--------------------+
mAP score for QC dataset: 0.7948302469135802
```

Figure 23: Example QC Outputs of YOLOv2 notebook *– An equivalent figure is shown to the user after playing section shown in fig. 14 for YOLOv2.* ***A)*** *The table shows the user the most important quality metrics numerically.* ***B)*** *The p-r curves for two classes investigated in this example (dividing or division complete).* ***C)*** *Visual depiction of model predictions versus the input and the target annotation.*

iv. fnet

In the fnet notebook, the evaluation of quality metrics follows closely the route of the CARE notebook, using the same image quality metrics as above (Fig. 24). The key difference is that only prediction vs. target errors are evaluated because the modality of the ground-truth, usually a label-free brightfield or EM image and the predicted output are too different to be evaluated by the values of pixel intensities which the NRMSE, PSNR and SSIM metrics are based on. In the example below, the QC cell shows an example of input, target, prediction, and the error maps and metrics between target and prediction. Hence, the QC section in fnet could almost be completely taken over from CARE, apart from the prediction step, the implementation of which differs significantly between both methods, and which is discussed in detail in the next section.

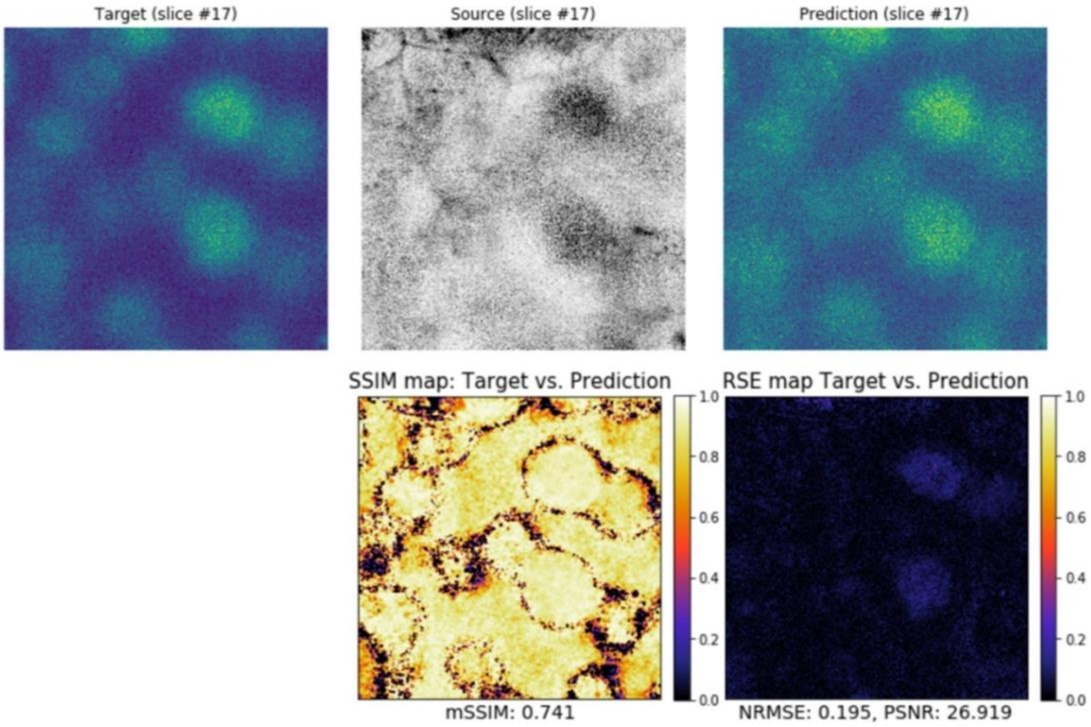


***Figure 24: Example QC Outputs of fnet 3D notebook** – An equivalent figure is shown to the user after playing the section shown in fig. 20 for fnet 3D: The same metrics are calculated as for Fig. 21, but in fnet no metric between input and target is calculated as images tend to lie in different domains (e.g. brightfield and fluorescence) which are not always commensurable.*

II. 2. 1. g. Execution of Prediction and saving results

Aims: In the prediction step, the previously trained models need to be useable to process and save unseen data for the user. The prediction step should therefore consist of three inputs by the user which should be sufficient to achieve this: The model to be used (*Prediction_model_path*), and if possible, which model weights (i.e. in YOLO, best validation loss or best mAP), the path of the raw data (*Data_folder*), and the path where the predicted data should be saved (*Results_folder*) (Fig. 25). After the prediction, the user should see at least one example of an input and prediction pair, as a minimal assurance that the predictions appear reasonable.

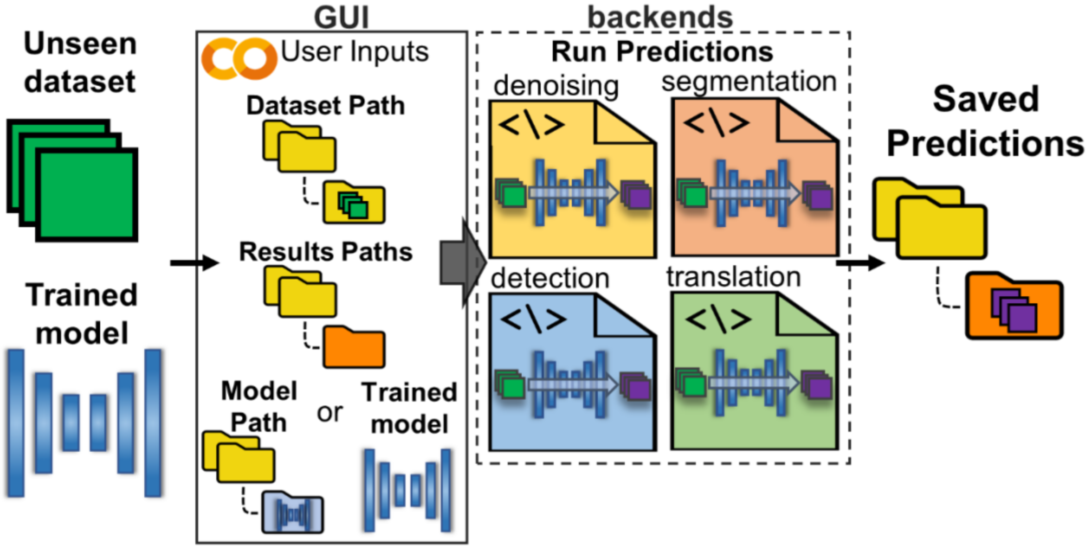


Figure 25 - Envisioned flow of step 7 – Executing Predictions on unseen data - *The user provides an unseen dataset and a trained model. In the GUI, the user gives the path to this dataset and the path to where predictions should be stored and which model to use, either from within the runtime or a previously saved model. The different backends perform predictions using their respective functions. All results are then saved in the 'Results_path' indicated in the GUI.*

The prediction section is consistent throughout notebooks and the three input arguments are equivalent throughout. The user first chooses a model folder, and as in the quality control section has the choice between using the model currently used in the session, i.e. trained in the runtime, via a checkbox or to use a previously trained model saved in the user's drive by inputting the path to the model folder in the input field in the GUI.

i. CARE 2D

In CARE, predictions are made after loading the model from the weights file saved in the model_folder using CARE's custom *CARE* class and leaving the configuration argument empty. The path to the model is deconstructed into the base directory and the daughter directory of the model and used as input arguments to the CARE function which creates the model from the weights file found under this path. Predictions are then performed on this object via the predict function on this CARE class model object. Because the predict function operates only on individual images, it needs to be run in a loop that cycles through the *Data_folder*. Since predicted images are not automatically saved, this is also done within the same loop (Fig. 26A – CARE 2D).

ii. U-Net 2D

In U-Net, the model is loaded via Keras' *load_model* function using the *Prediction_model_folder* as input. Predictions are performed using a custom prediction function designed for the ZeroCostDL4Mic notebook. This function uses as input arguments simply the *Data_folder* and the previously loaded *unet* model object. Here, the predictions are appended to a list saved in the *Results_folder* via a custom *saveResults* function created for this notebook (Fig. 26A – U-Net 2D).

iii. YOLOv2

In YOLOv2, several significant changes were made to the repository's default prediction function. This was deemed necessary because the original outputs of the prediction function were images with bounding boxes drawn on to the image together with the class names and confidence levels for each bounding box. These outputs are nearly unusable for bioimage analysis as these images contain neither the coordinates of the bounding boxes nor are these images editable, allowing the user to edit predicted bounding boxes and fine-tune or remove any predictions.

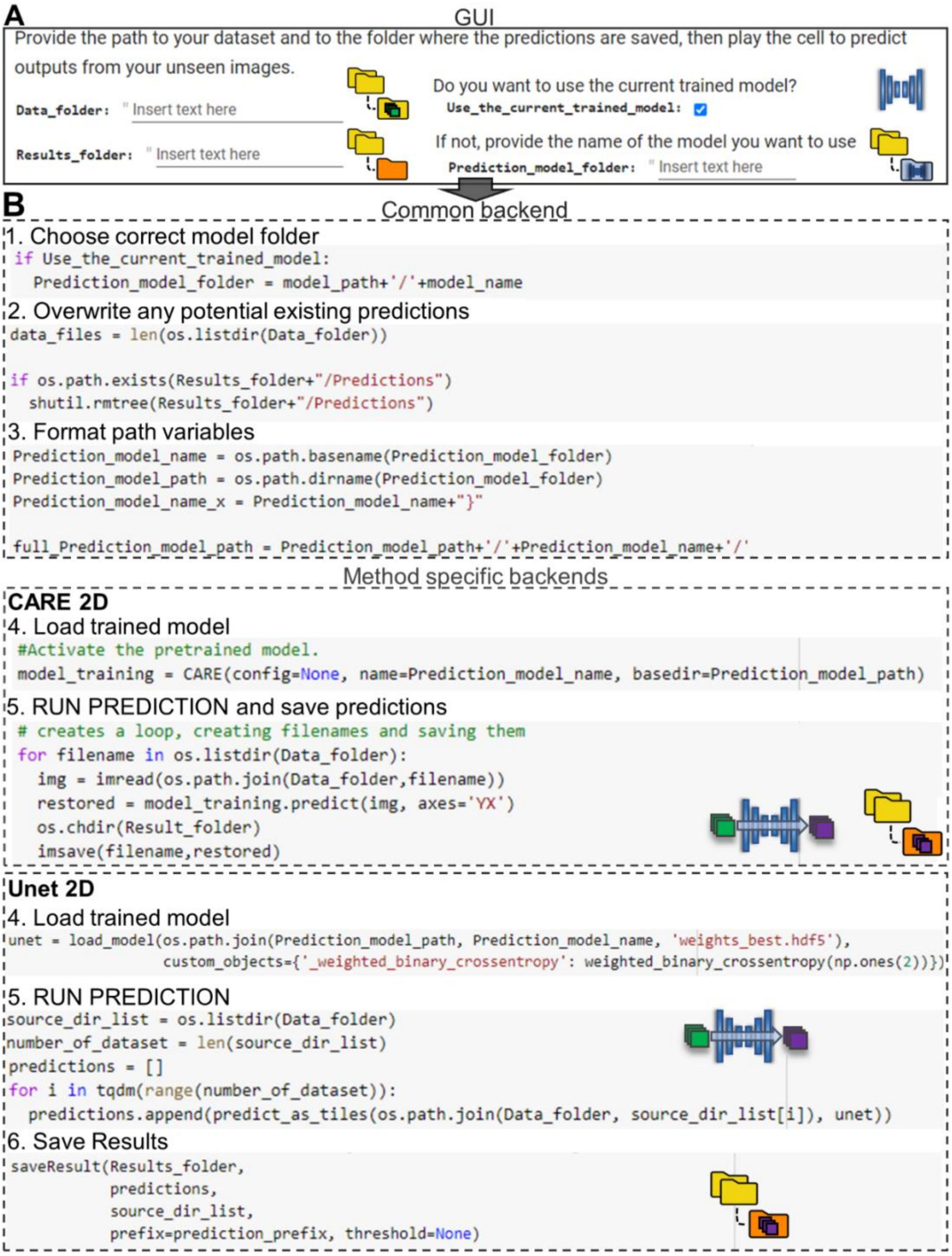
Hence, I created a new prediction function with several sub-functions that would not only save the predicted images but would also create a list of all bounding boxes, their predicted classes, and confidence levels and save them in a .csv file which can be read, for example, into ImageJ's or Fiji's ROI manager and thus viewed and edited for bioimage analysis.

In the YOLOv2 predictions are again performed in a loop because the prediction function of the YOLO class, based on Keras, only predicts on individual images. However, in the YOLOv2 notebook, this led to significant memory leakage, which often overloaded the notebook's RAM. Therefore, I added an additional line to the prediction loop, which would clear the session's memory after each prediction. This allows the notebook to run smoothly and perform hundreds to thousands of predictions without crashing (Fig. 26B).

iv. fnet

To run predictions in fnet, the paths to the files to be predicted need to be inserted into a .csv file as in training. We therefore first need to create a .csv file in the Prediction cell of the notebook using the files in Data_folder to fill the rows. Next, we need to edit the predict.sh script (or predict_2D.sh for fnet 2D) which needs to be filled with the user's inputs on the *Prediction_model_folder* and the path to the csv file. Furthermore, the prediction file assumes that the data folder contains training files with targets. To avoid the fnet from searching for these files in an additional .csv file which we do not need to create, we need to remove this line from the script to avoid errors. All these modifications are done using shell search commands, similar to those used in the installation cell.

By default, fnet creates a Prediction folder in the *Results_folder*, which is filled with one folder for each image each of which contains three images, the input, the prediction, and a 'target'. However, it is not clear from the fnet annotation why there is a target in the prediction folder. These files do not contain the original file name and are simply named 00, 01, 02, etc. To bring the results into a more intuitive folder structure, the predictions from each folder created in this way by fnet are copied into the user's *Results_folder* under the original filename with the suffix *predicted.* After all files are copied in this way, the fnet-created *Prediction* folder is deleted, leaving the user with outputs in a similar format as in the other notebooks with no additional inputs (Fig. 26B).



Provide the path to your dataset and to the folder where the predictions are saved, then play the cell to predict outputs from your unseen images.

Data_folder: " Insert text here

Results_folder: " Insert text here

Do you want to use the current trained model?

Use_the_current_trained_model: ☑

If not, provide the name of the model you want to use

Prediction_model_folder: " Insert text here

1. Choose correct model folder

```
if Use_the_current_trained_model:
  Prediction_model_folder = model_path+'/'+model_name
```

2. Overwrite any potential existing predictions

```
data_files = len(os.listdir(Data_folder))

if os.path.exists(Results_folder+"/Predictions")
  shutil.rmtree(Results_folder+"/Predictions")
```

3. Format path variables

```
Prediction_model_name = os.path.basename(Prediction_model_folder)
Prediction_model_path = os.path.dirname(Prediction_model_folder)
Prediction_model_name_x = Prediction_model_name+"}"

full_Prediction_model_path = Prediction_model_path+'/'+Prediction_model_name+'/'
```

CARE 2D

4. Load trained model

```
#Activate the pretrained model.
model_training = CARE(config=None, name=Prediction_model_name, basedir=Prediction_model_path)
```

5. RUN PREDICTION and save predictions

```
# creates a loop, creating filenames and saving them
for filename in os.listdir(Data_folder):
  img = imread(os.path.join(Data_folder,filename))
  restored = model_training.predict(img, axes='YX')
  os.chdir(Result_folder)
  imsave(filename,restored)
```

Unet 2D

4. Load trained model

```
unet = load_model(os.path.join(Prediction_model_path, Prediction_model_name, 'weights_best.hdf5'),
                  custom_objects={'_weighted_binary_crossentropy': weighted_binary_crossentropy(np.ones(2))})
```

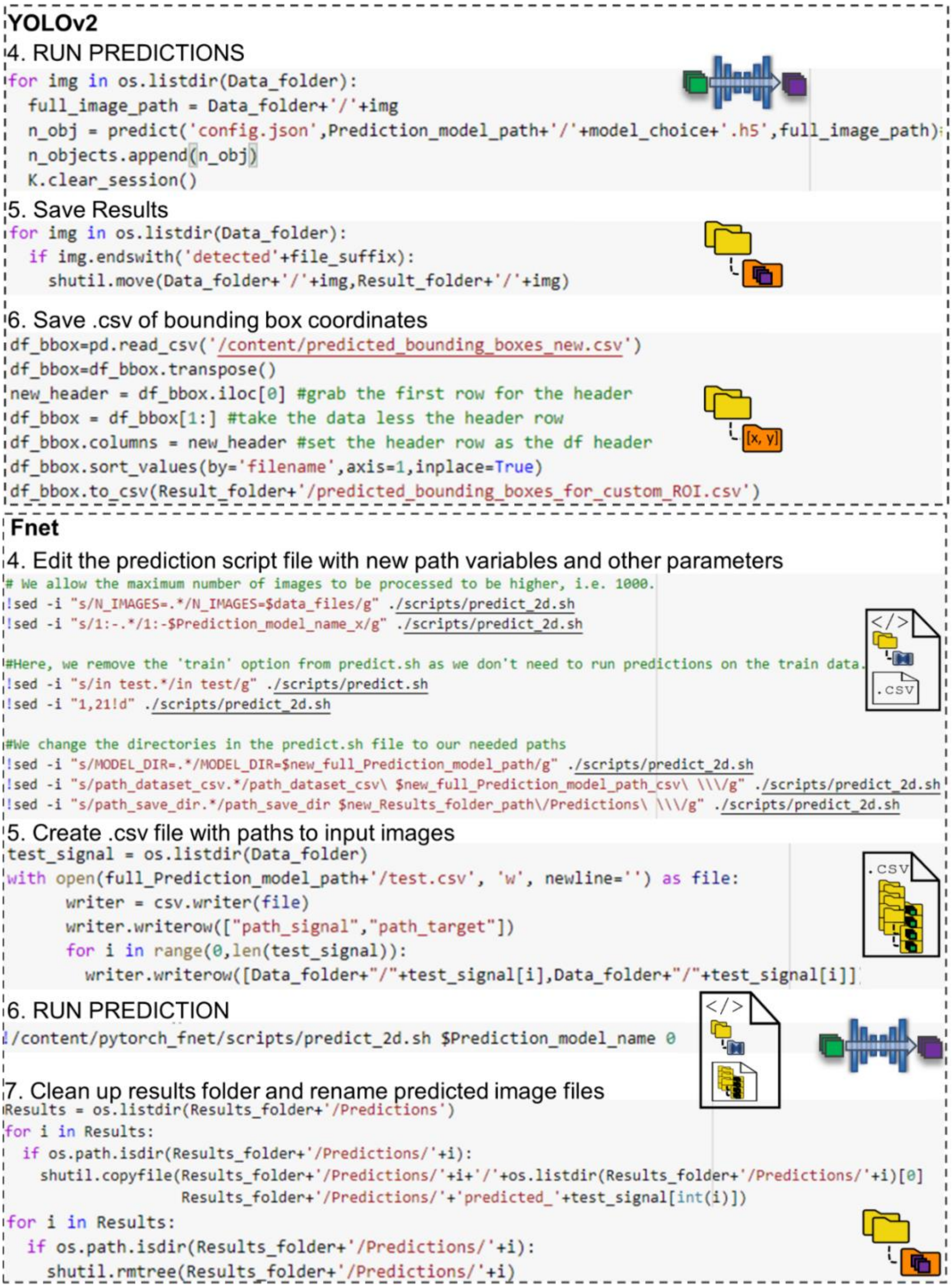
5. RUN PREDICTION

```
source_dir_list = os.listdir(Data_folder)
number_of_dataset = len(source_dir_list)
predictions = []
for i in tqdm(range(number_of_dataset)):
  predictions.append(predict_as_tiles(os.path.join(Data_folder, source_dir_list[i]), unet))
```

6. Save Results

```
saveResult(Results_folder,
           predictions,
           source_dir_list,
           prefix=prediction_prefix, threshold=None)
```

Figure 26A - Step 7 – part 1 – Executing prediction on unseen datasets: CARE and U-Net 2D *- A) Common GUI for all notebooks.* ***B)*** *Common and method-specific backends of the DL-methods in the notebooks for CARE and fnet.*



YOLOv2

4. RUN PREDICTIONS

```
for img in os.listdir(Data_folder):
  full_image_path = Data_folder+'/'+img
  n_obj = predict('config.json',Prediction_model_path+'/'+model_choice+'.h5',full_image_path)
  n_objects.append(n_obj)
  K.clear_session()
```

5. Save Results

```
for img in os.listdir(Data_folder):
  if img.endswith('detected'+file_suffix):
    shutil.move(Data_folder+'/'+img,Result_folder+'/'+img)
```

6. Save .csv of bounding box coordinates

```
df_bbox=pd.read_csv('/content/predicted_bounding_boxes_new.csv')
df_bbox=df_bbox.transpose()
new_header = df_bbox.iloc[0] #grab the first row for the header
df_bbox = df_bbox[1:] #take the data less the header row
df_bbox.columns = new_header #set the header row as the df header
df_bbox.sort_values(by='filename',axis=1,inplace=True)
df_bbox.to_csv(Result_folder+'/predicted_bounding_boxes_for_custom_ROI.csv')
```

Fnet

4. Edit the prediction script file with new path variables and other parameters

```
# We allow the maximum number of images to be processed to be higher, i.e. 1000.
!sed -i "s/N_IMAGES=.*/N_IMAGES=$data_files/g" ./scripts/predict_2d.sh
!sed -i "s/1:-.*/1:-$Prediction_model_name_x/g" ./scripts/predict_2d.sh

#Here, we remove the 'train' option from predict.sh as we don't need to run predictions on the train data.
!sed -i "s/in test.*/in test/g" ./scripts/predict.sh
!sed -i "1,21!d" ./scripts/predict_2d.sh

#We change the directories in the predict.sh file to our needed paths
!sed -i "s/MODEL_DIR=.*/MODEL_DIR=$new_full_Prediction_model_path/g" ./scripts/predict_2d.sh
!sed -i "s/path_dataset_csv.*/path_dataset_csv\ $new_full_Prediction_model_path_csv\ \\\/g" ./scripts/predict_2d.sh
!sed -i "s/path_save_dir.*/path_save_dir $new_Results_folder_path\/Predictions\ \\\/g" ./scripts/predict_2d.sh
```

5. Create .csv file with paths to input images

```
test_signal = os.listdir(Data_folder)
with open(full_Prediction_model_path+'/test.csv', 'w', newline='') as file:
      writer = csv.writer(file)
      writer.writerow(["path_signal","path_target"])
      for i in range(0,len(test_signal)):
        writer.writerow([Data_folder+"/"+test_signal[i],Data_folder+"/"+test_signal[i]]
```

6. RUN PREDICTION

```
!/content/pytorch_fnet/scripts/predict_2d.sh $Prediction_model_name 0
```

7. Clean up results folder and rename predicted image files

```
Results = os.listdir(Results_folder+'/Predictions')
for i in Results:
  if os.path.isdir(Results_folder+'/Predictions/'+i):
    shutil.copyfile(Results_folder+'/Predictions/'+i+'/'+os.listdir(Results_folder+'/Predictions/'+i)[0]
                    Results_folder+'/Predictions/'+'predicted_'+test_signal[int(i)])
for i in Results:
  if os.path.isdir(Results_folder+'/Predictions/'+i):
    shutil.rmtree(Results_folder+'/Predictions/'+i)
```

Figure 26 B - Step 7 – part 2 – Executing prediction on unseen datasets: YOLOv2 and fnet - *Continuation of previous figure showing the backends of the YOLOv2 and U-Net notebooks' prediction cells.*

II. 2. 2. Auxiliary steps to overcome barriers of DL in bioimage analysis

With the implementation of the above steps into a GUI built in Colab, we created a core pipeline that allows users to easily apply different DL-tools without significant interaction with a code environment. This directly addressed four key problems that limit access to DL: the lack of access to inexpensive hardware acceleration, the lack of technical knowledge of coding, the lack of tools useable by non-developers, and the absence of simple quality controls for DL methods. However, these core steps could be enhanced to address a further issue: the difficulty in obtaining large, curated datasets required to train DL-methods. In ZeroCostDL4Mic, we can at least alleviate the need for very large datasets by providing access to data augmentation and transfer learning within the workflow of the notebook.

II. 2. 2. a. Data augmentation

Aims: One of the barriers to using DL-tools is the limited access to large enough datasets with paired images or image-label pairs, as these are time-intensive in curation. One way in which this problem can be mitigated is by augmenting data through image transformations. Several augmentation packages exist for Python. However, like DL-tools themselves, augmentation pipelines developed for DL often require an understanding of code, limiting access to this potentially performance-enhancing step. Here, we wanted to facilitate the augmentation by implementing it in a code-free form, allowing the user to make various choices for the augmentation.

As inputs, this cell should simply take the datasets which the user originally chose in Section 1 of the notebook. The augmented images can be either saved in the notebook or used for training without saving. However, if saved, they should not be mixed with the original dataset to prevent this dataset from changing across different model training sessions. The augmentation cell needs to be optional since users may not want to enhance their dataset artificially. Therefore, this step must include an option for users to skip the step entirely or to actively choose that no augmentation is desired.

In the ZeroCostDL4Mic notebooks, the augmentation steps follow immediately after the choice of the main training parameters.

i. Making augmentation optional

To make the augmentation cell optional, the first possible user input is a tick-box that gives the user the option to select the use of augmentation or opt-out. The choice will determine the value of a Boolean variable (True if selected and False if un-selected). By default, the augmentation choice Boolean is set to False at the end of the previous cell (the training parameter choice, see Fig. 12-15). Defining the variable before the augmentation cell allows the user to skip the augmentation cell altogether, without leading to later VariableErrors when the augmentation choice Boolean is called in the code.

ii. Augmentation Options

Next, the augmentation box includes variable adjustable parameters which are represented by sliders or dropdown options, allowing interaction without requiring the user to type in the code. By using sliders, we limit the choice and number of possible augmentations. This prevents users from inserting unsuitable values that could lead to datasets too large for the Colab environment or might result in redundant augmentations, i.e., rotating an image twice by 180 degrees.

Implementing augmentations in a standardised manner was challenging, as the datasets to be augmented may vary in data type, dimensionality, and size. Furthermore, not all augmentations may be equally suitable for different functionalities of the DL methods. For example, an augmentation that adds noise or blur to a denoising dataset might introduce biases into the images, making the network’s performance on a real dataset worse. In biological contexts, it is important to choose augmentations that lead to instances of images mimicking the original data. For example, networks that learn the shapes of a certain cell-type might not benefit from augmentations that significantly change the cells' shapes, e.g. under shearing augmentations.

The next most conservative transformation for biological data is translation. The image is shifted in a specific direction, and the rest is filled either by mirroring the remaining content or leaving it blank. Using translations, images can be arbitrarily augmented. However, this approach is limited by how far a shift is needed to constitute a significant change.

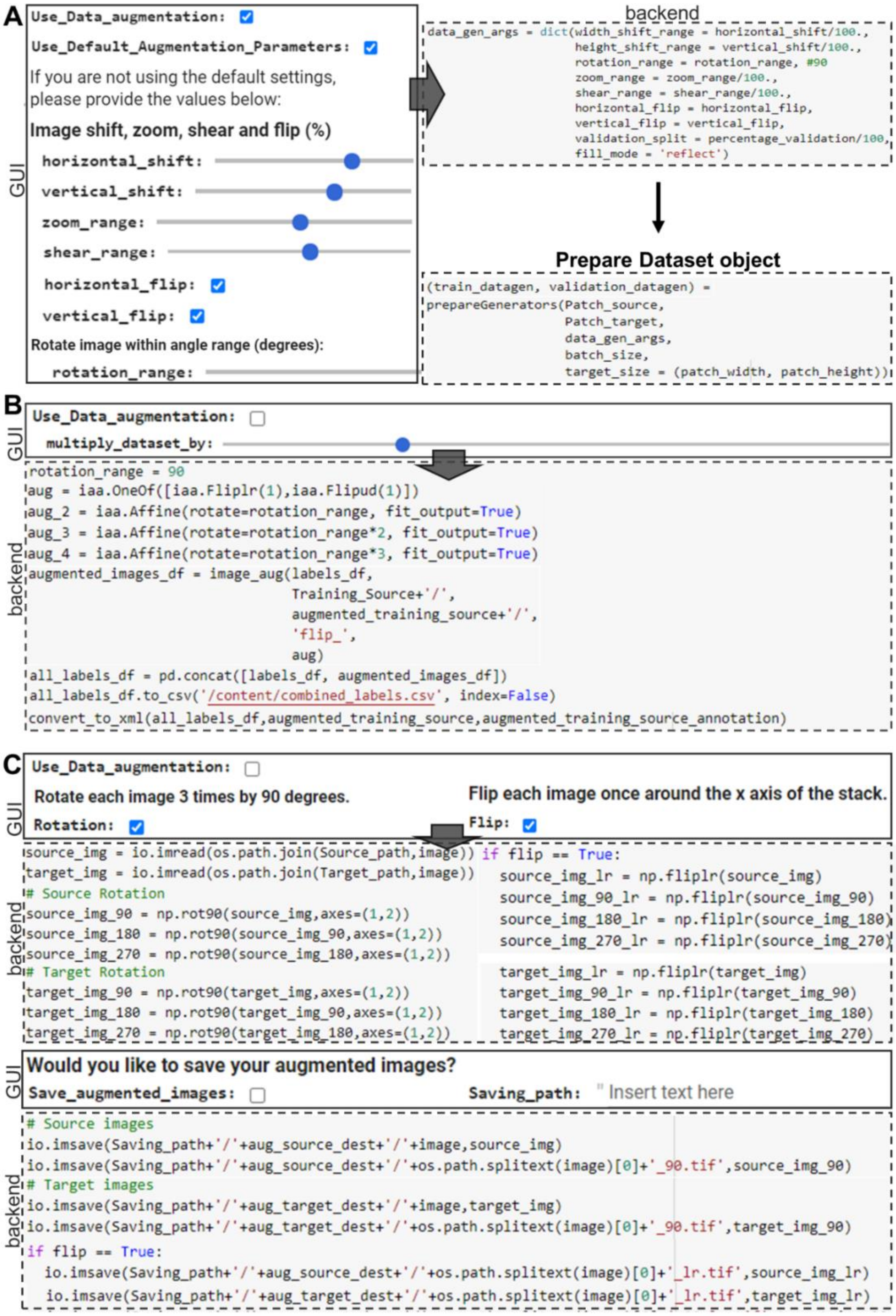
Given these considerations on augmentation type and dataset, the augmentation options provided in each notebook differ slightly between methods. However, there are augmentations that are likely to be almost universally useful for bioimaging and included in all notebooks. All notebooks provide at least rotation and mirroring (flip) augmentations, since from an image information standpoint, many bioimaging datasets can be assumed to be orientationally invariant (i.e. whether a cell is 'upside-down' or 'left-to-right-flipped' should not change the information content of the image) (Fig. 27A). Hence, these geometric augmentations can be interpreted as relatively conservative as they do not change the shape of the objects on the images but only their orientation representing realistic variations in biological data. Geometric augmentations can be easily implemented on NumPy[174] arrays using matrix transformations and easily extended to 3D datasets that many existing augmentation pipelines do not consider (Fig. 27C). Geometric augmentation by rotation around 90 degrees angles is the simplest form of augmentation as no parts of the images need to be cut-off by the image edges. Using ninety-degree rotations and mirroring can increase a dataset at best by a factor of 8, since every single image can be quadrupled by rotations around 90 degrees and again doubled by flipping each image once around either the x or the y axis, without creating any redundant images. These augmentations were performed on datasets used in the 3D methods, such as fnet and CARE 3D. The user can choose whether one or both types of augmentations should be performed on the data in these notebooks.

A different type of augmentation is implemented in YOLOv2 and CARE 2D. In both notebooks, augmentations are executed by dedicated Python packages. In YOLOv2, we use the img_aug Python package, which can perform geometric augmentations on images and bounding boxes (Fig. 27B). However, the img_aug library may be able to calculate the new coordinates for each bounding box after rotation or flipping but will not automatically save this information in new annotation files, which are necessary to train the YOLOv2 network. Hence, I created a function that converts the new bounding box coordinates for the geometrically augmented images into PASCAL VOC style .xml files which are used to train the YOLOv2 model. These files are saved in additional folders that are by default saved in the /content folder but can optionally be saved under a path of the user's choice.

Similarly, in CARE 2D, the Augmentor[196] package is used for augmentations and was implemented by Guillaume Jacquemet. Here, the augmented images are also saved in a new location and then used as the new *Training_Source* and *Training_Target* to train the neural network (not shown).

A more elegant solution for augmentation can be applied in the purely Keras-based U-Net 2D notebook. The Keras data generator has an inbuilt augmentation argument which can be filled with many different augmentation choices. In the augmentation cell, the user's augmentation choices are simply put into the augmentation library and passed to the data generator in the next cell (Fig. 28A). Using this form of augmentation means the augmented images are not by default stored in the runtime and cannot overload the notebook's memory. This has the significant advantage that datasets can theoretically be augmented infinitely, despite the memory limitations in Colab.

Given the various options for augmentations and different requirements per data type and the backend of the methods, the augmentation cell is the most variable cell in the project (see Fig. 27). However, the interactive elements of the GUI are maintained throughout, which provides a minimum of the continuity desired in the project.



Use_Data_augmentation: ☑

Use_Default_Augmentation_Parameters: ☑

If you are not using the default settings, please provide the values below:

Image shift, zoom, shear and flip (%)

horizontal_shift: ●

vertical_shift: ●

zoom_range: ●

shear_range: ●

horizontal_flip: ☑

vertical_flip: ☑

Rotate image within angle range (degrees):

rotation_range:

```
data_gen_args = dict(width_shift_range = horizontal_shift/100.,
                     height_shift_range = vertical_shift/100.,
                     rotation_range = rotation_range, #90
                     zoom_range = zoom_range/100.,
                     shear_range = shear_range/100.,
                     horizontal_flip = horizontal_flip,
                     vertical_flip = vertical_flip,
                     validation_split = percentage_validation/100,
                     fill_mode = 'reflect')
```

```
(train_datagen, validation_datagen) =
prepareGenerators(Patch_source,
                  Patch_target,
                  data_gen_args,
                  batch_size,
                  target_size = (patch_width, patch_height))
```

Use_Data_augmentation: □

multiply_dataset_by: ●

```
rotation_range = 90
aug = iaa.OneOf([iaa.Fliplr(1),iaa.Flipud(1)])
aug_2 = iaa.Affine(rotate=rotation_range, fit_output=True)
aug_3 = iaa.Affine(rotate=rotation_range*2, fit_output=True)
aug_4 = iaa.Affine(rotate=rotation_range*3, fit_output=True)
augmented_images_df = image_aug(labels_df,
                                Training_Source+'/',
                                augmented_training_source+'/',
                                'flip_',
                                aug)
all_labels_df = pd.concat([labels_df, augmented_images_df])
all_labels_df.to_csv('/content/combined_labels.csv', index=False)
convert_to_xml(all_labels_df,augmented_training_source,augmented_training_source_annotation)
```

Use_Data_augmentation: □

Rotate each image 3 times by 90 degrees.

Rotation: ☑

Flip each image once around the x axis of the stack.

Flip: ☑

```
source_img = io.imread(os.path.join(Source_path,image))
target_img = io.imread(os.path.join(Target_path,image))
# Source Rotation
source_img_90 = np.rot90(source_img,axes=(1,2))
source_img_180 = np.rot90(source_img_90,axes=(1,2))
source_img_270 = np.rot90(source_img_180,axes=(1,2))
# Target Rotation
target_img_90 = np.rot90(target_img,axes=(1,2))
target_img_180 = np.rot90(target_img_90,axes=(1,2))
target_img_270 = np.rot90(target_img_180,axes=(1,2))
```

```
if flip == True:
  source_img_lr = np.fliplr(source_img)
  source_img_90_lr = np.fliplr(source_img_90)
  source_img_180_lr = np.fliplr(source_img_180)
  source_img_270_lr = np.fliplr(source_img_270)

  target_img_lr = np.fliplr(target_img)
  target_img_90_lr = np.fliplr(target_img_90)
  target_img_180_lr = np.fliplr(target_img_180)
  target_img_270_lr = np.fliplr(target_img_270)
```

Would you like to save your augmented images?

Save_augmented_images: □

Saving_path: " Insert text here

```
# Source images
io.imsave(Saving_path+'/'+aug_source_dest+'/'+image,source_img)
io.imsave(Saving_path+'/'+aug_source_dest+'/'+os.path.splitext(image)[0]+'_90.tif',source_img_90)
# Target images
io.imsave(Saving_path+'/'+aug_target_dest+'/'+image,target_img)
io.imsave(Saving_path+'/'+aug_target_dest+'/'+os.path.splitext(image)[0]+'_90.tif',target_img_90)
if flip == True:
  io.imsave(Saving_path+'/'+aug_source_dest+'/'+os.path.splitext(image)[0]+'_lr.tif',source_img_lr)
  io.imsave(Saving_path+'/'+aug_target_dest+'/'+os.path.splitext(image)[0]+'_lr.tif',target_img_lr)
```

Figure 27 - Implementing Augmentation in different notebooks - A) *Augmentation in U-Net using* Keras *datagenerator B) Augmentation in YOLOv2 using the img_aug package (here, for simplicity, only one augmentation is shown which is repeated in the notebook depending on the multiplication chosen by the user. In each round of augmentation, the new annotations are saved in the combined_labels.csv C) Augmentation in fnet, using* NumPy *array functions.*

II. 2. 2. b. Transfer learning

Aims: Using a pretrained model for further training has several important advantages. Firstly, a limitation of the Colab environment is its limited runtime, which limits the number of epochs a model can be trained for, imposing an upper limit on any convergence achievable during training and ultimately limiting the model's performance. This also means that models need to be retrained from randomized weights even when adjusting parameters only slightly, taking significantly longer than using a model with weights already initialized for a task. Therefore, it is beneficial to include the option to load models with existing weights into the runtime and retrain these models in the notebooks. This would also allow users to exploit the vast array of available pre-trained models often published by developer groups and trained on much larger datasets than would be feasible for use in Google Colab. This builds a useful connection point for the community creating trained DL-model repositories, such as bioimage.io.

The simplest way to implement this is by giving the user the ability to choose the path to a pre-trained model folder containing the model weights file to be used. This model should be chosen after the training parameters and the *model_path* and *model_name* parameters are chosen in Section 1. This is necessary to clarify where the retrained model file should be saved and to avoid overwriting the pre-trained model weights. Transfer learning should be an optional choice, and the notebook needs to run error-free with or without the transfer learning option.

In the ZeroCostDL4Mic notebooks, transfer learning is not named as such. The step is instead referred to as '*Using weights from a pre-trained model as initial weights'.* This change was made with the user-base in mind, which we assume to be primarily biologists with no prior knowledge of machine learning jargon which *transfer learning* is part of. This section is consistent throughout most of the notebooks since the steps in backend of the notebook are usually very similar.

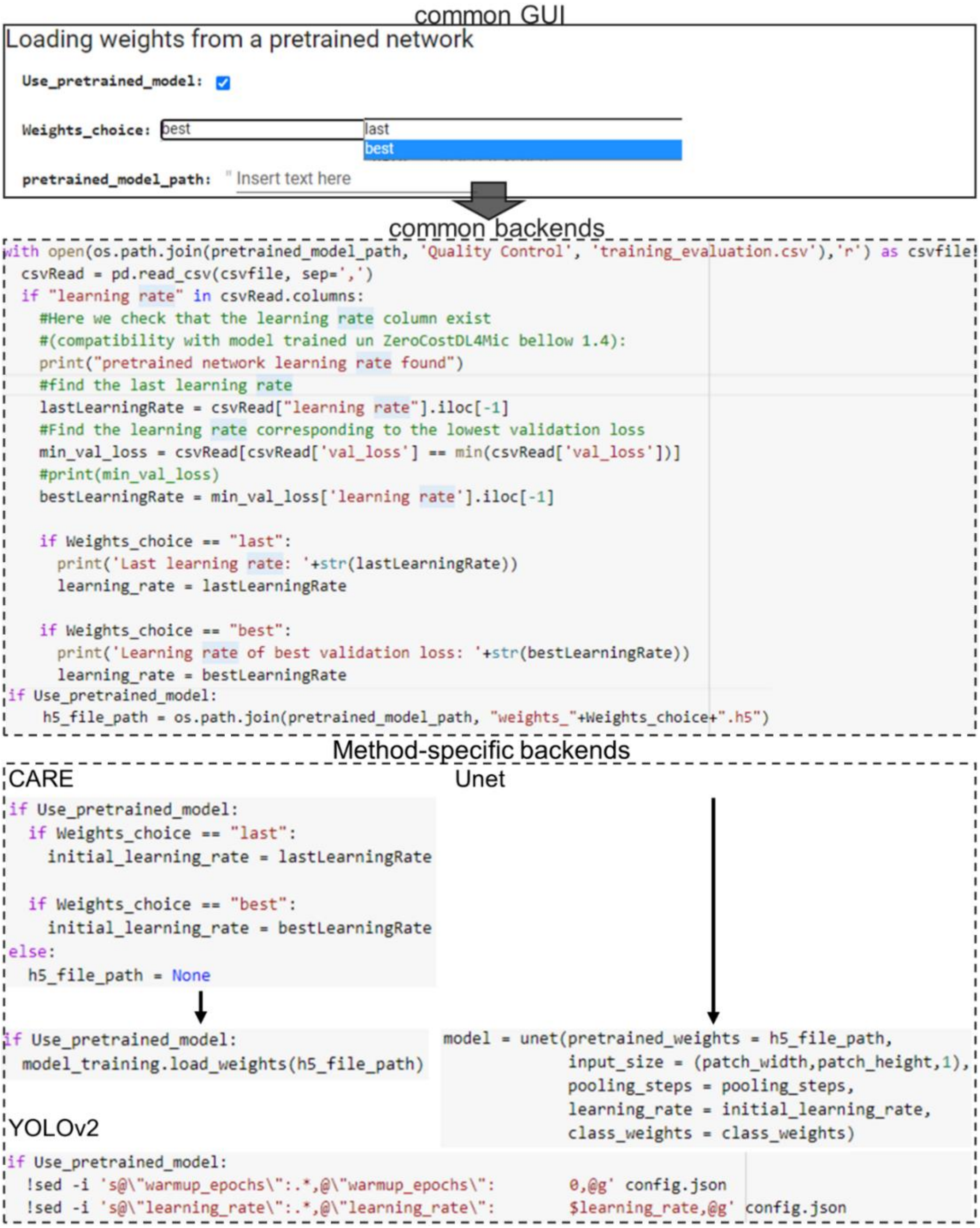
i. Using pretrained models in CARE, U-Net 2D and YOLOv2

First, the user chooses, using a tick-box, whether to use a pre-trained model for training or not. This tick-box is similarly designed to the one in the augmentation cell and represents a Boolean which is set by default to 'False' (Fig. 28 top).

Next, the user can enter the path to a previously trained model into the input box named, *'Pretrained_model_path'*. A further option which is available only in some methods (StarDist, CellPose) is to use pretrained models provided with the method itself. In these cases, to provide the user a choice between using their own model or a pre-trained model from within the method, the cell contains an additional dropdown menu containing the different trained model names and the option to use a 'Model_from_file' after which the user enters the *Pretrained_model_path* as above (Fig. 28 top). If the model is trained from a previously trained model on the user's google drive, it may contain different weight files, such as the '*last_weights*' and '*best_weights'*. In these cases, the cell contains an additional dropdown option, in which the respective model weights can be selected (e.g. YOLOv2, CARE).

The different methods handle pre-trained weights slightly differently. In CARE, the pretrained model weights can be directly loaded into the model object after its creation in the *Create dataset and model objects* cell. In both U-Net and YOLOv2, the option to use pretrained weights is used by default if a path to a pre-trained model is given. In U-Net, the parameter, *pretrained_model_path* exists in the data generator function. If given as 'None' the network uses a blank model to start training. Otherwise, the *Pretrained_model_path* is used with the weights file chosen by the user to start training from the pre-trained model. In YOLOv2, the *Pretrained_model_path* needs to be added to the *config.json* and will then be automatically used for training. This is done like the other edits to the *config.json* via shell search function (see e.g. Fig. 14, 15). If a pre-trained model path does not exist, the network will be automatically trained from the default model weights. In YOLOv2, an additional feature needs to be considered in the form of warm-up epochs. Using default settings, the default model weights are 'adjusted' to the user's data set by training only the top layers of the network for 3 epochs, which is known to prevent the model from quickly reaching a vanishing gradient in a new dataset. When using pre-trained weights, this 'warm-up' is not necessary, and this parameter should be set to 0.

In notebooks which adjust the learning rate, we also need to ensure that any further training continues at a sufficiently low learning rate to prevent the model from diverging from an optimal loss. To find this learning rate, a *training_losses.csv* file is searched for in the *Pretrained_model_folder*. If the model was trained in ZeroCostDL4Mic, this file is automatically created during training. This losses.csv file contains the learning rate and losses for each epoch of the previously trained model. The learning rate corresponding to the epoch with the lowest validation loss is used to continue training the model, since generally the model weights yielding the lowest validation loss are saved in the ZeroCostDL4Mic notebooks. Where other weight files are saved, e.g. in YOLOv2 which saves the weights for the highest mAP score as well (see above), the learning rate corresponding to the highest mAP score achieved during training is used.



Loading weights from a pretrained network

Use_pretrained_model: ☑

Weights_choice: best (last / best)

pretrained_model_path: " Insert text here

```
with open(os.path.join(pretrained_model_path, 'Quality Control', 'training_evaluation.csv'),'r') as csvfile:
  csvRead = pd.read_csv(csvfile, sep=',')
  if "learning rate" in csvRead.columns:
    #Here we check that the learning rate column exist
    #(compatibility with model trained un ZeroCostDL4Mic bellow 1.4):
    print("pretrained network learning rate found")
    #find the last learning rate
    lastLearningRate = csvRead["learning rate"].iloc[-1]
    #Find the learning rate corresponding to the lowest validation loss
    min_val_loss = csvRead[csvRead['val_loss'] == min(csvRead['val_loss'])]
    #print(min_val_loss)
    bestLearningRate = min_val_loss['learning rate'].iloc[-1]

    if Weights_choice == "last":
      print('Last learning rate: '+str(lastLearningRate))
      learning_rate = lastLearningRate

    if Weights_choice == "best":
      print('Learning rate of best validation loss: '+str(bestLearningRate))
      learning_rate = bestLearningRate
if Use_pretrained_model:
    h5_file_path = os.path.join(pretrained_model_path, "weights_"+Weights_choice+".h5")
```

CARE

```
if Use_pretrained_model:
  if Weights_choice == "last":
    initial_learning_rate = lastLearningRate

  if Weights_choice == "best":
    initial_learning_rate = bestLearningRate
else:
  h5_file_path = None
```

```
if Use_pretrained_model:
  model_training.load_weights(h5_file_path)
```

Unet

```
model = unet(pretrained_weights = h5_file_path,
             input_size = (patch_width,patch_height,1),
             pooling_steps = pooling_steps,
             learning_rate = initial_learning_rate,
             class_weights = class_weights)
```

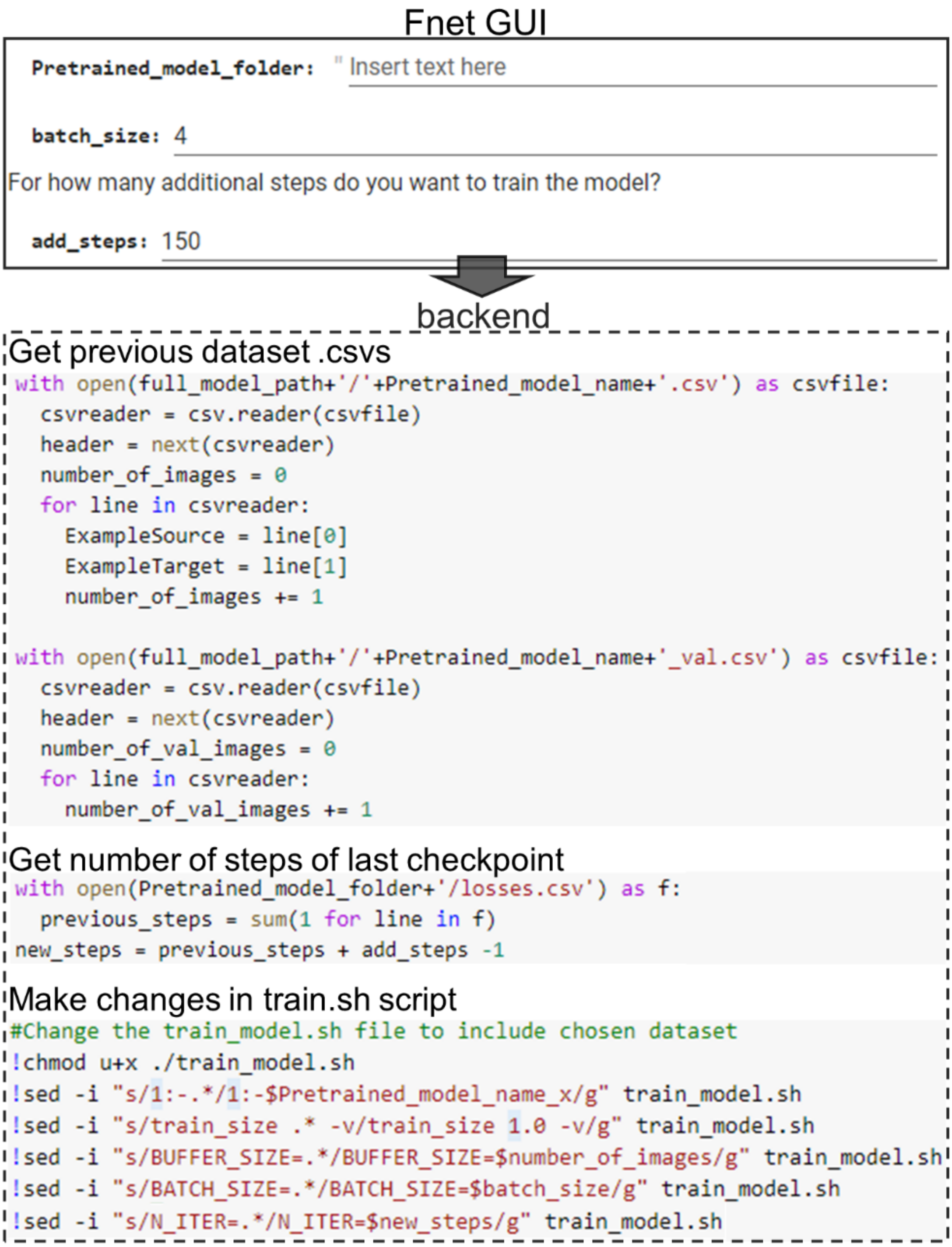
YOLOv2

```
if Use_pretrained_model:
  !sed -i 's@\"warmup_epochs\":.*,@\"warmup_epochs\":        0,@g' config.json
  !sed -i 's@\"learning_rate\":.*,@\"learning_rate\":        $learning_rate,@g' config.json
```

***Figure 28: Implementing Transfer Learning in CARE, U-Net and YOLOv2 notebooks** - shown is how the learning rate for the model is determined when re-trained by using the originally trained models' training_evaluation.csv file which is created during training and contains losses and learning rates for each epoch. In CARE, different weight files are accessible which can be chosen from in CARE's GUI. In U-Net, the weight file detected above is directly inserted into the unet model function which instantiates the model object as in Fig.13. In YOLOv2, the primary step is changing the warm-up epochs and learning rate in the 'config.json'. The 'model_folder' is changed by default in the 'config.json' in YOLOv2. This is therefore not specific to the Transfer learning section.*

ii. Using pretrained models in fnet

An exception to the transfer learning section in the ZeroCostDL4Mic notebooks is the fnet notebook. Here, the section is designed to allow continued training on a model using the same filename and file location (Fig. 29). Hence, if the user opts to use a pre-trained model, they will automatically continue training from a specific weight file that will then be overwritten. The reason for this is that in the Colab environment, fnet fits very slowly to the training datasets, which regularly results in the notebook's maximum runtime being exceeded. Hence, the most likely use-case for the transfer learning section in the fnet notebooks during development of the notebook and its testing was to continue training from the last saved checkpoint and finish training on the exact same dataset. In a future version, the notebook might be adapted to the more general use-case that already applies to the other notebooks. A difficulty in fnet is that training already requires several different files to be edited and named in a very specific way for the model to be trained error-free. The adaptation of the notebook into a consistent format with the other notebooks would thus require significant effort which was not yet feasible within the timeframe of this project.



Fnet GUI

Pretrained_model_folder: " Insert text here

batch_size: 4

For how many additional steps do you want to train the model?

add_steps: 150

backend

Get previous dataset .csvs

```
with open(full_model_path+'/'+Pretrained_model_name+'.csv') as csvfile:
  csvreader = csv.reader(csvfile)
  header = next(csvreader)
  number_of_images = 0
  for line in csvreader:
    ExampleSource = line[0]
    ExampleTarget = line[1]
    number_of_images += 1

with open(full_model_path+'/'+Pretrained_model_name+'_val.csv') as csvfile:
  csvreader = csv.reader(csvfile)
  header = next(csvreader)
  number_of_val_images = 0
  for line in csvreader:
    number_of_val_images += 1
```

Get number of steps of last checkpoint

```
with open(Pretrained_model_folder+'/losses.csv') as f:
  previous_steps = sum(1 for line in f)
new_steps = previous_steps + add_steps -1
```

Make changes in train.sh script

```
#Change the train_model.sh file to include chosen dataset
!chmod u+x ./train_model.sh
!sed -i "s/1:-.*/1:-$Pretrained_model_name_x/g" train_model.sh
!sed -i "s/train_size .* -v/train_size 1.0 -v/g" train_model.sh
!sed -i "s/BUFFER_SIZE=.*/BUFFER_SIZE=$number_of_images/g" train_model.sh
!sed -i "s/BATCH_SIZE=.*/BATCH_SIZE=$batch_size/g" train_model.sh
!sed -i "s/N_ITER=.*/N_ITER=$new_steps/g" train_model.sh
```

Figure 29: Implementing Transfer Learning in fnet notebook *– In fnet, the datasets used for the pretrained model are located via the path csv files created for the model's training. Training continues after the last checkpoint saved in the losses.csv file. The users' additional steps are added. The changed parameters are added to the 'train_model.sh' script before retraining.*

II. 2. 3. Summary of the complete ZeroCostDL4Mic notebook

After establishing a workflow and implementing a method in a ZeroCostDL4Mic, the user can now with relative ease and without coding train a neural network on a custom dataset. The workflow is summarised in the supplementary videos which are part of the publication, and which was created by Romain Laine. To summarise this briefly:

1. The user opens the notebook, is introduced to the method, and can read instructions on curating the dataset for the respective method.
2. The user then installs the method by playing the *Install Method and Dependencies* cell. This will automatically download all necessary packages into the Colab runtime, if they are not already native to Colab (e.g. for pure Keras methods). The notebook will then restart to ensure that any software versions that need to be updated are used correctly. The user then imports all the necessary packages and classes into the runtime by playing the cell *Import Dependencies*. This will also create all the function definitions for the notebook, so that training, re-training, quality control, and prediction can be played independently.
3. After following the instructions on dataset curation, users can upload their datasets to their Google Drive. When the datasets are in the user's Google Drive, they can access these datasets by mounting their google drive to the runtime, which is done by playing the *Mount Google Drive* cell.
4. Once the cell is connected, the user can access the paths to their training datasets and enter them into the GUI of the parameter input cell. The user also chooses a name for their model, and the path where the trained model weights should be saved once training completes. Other parameters such as the validation split, the learning rate and parameters specific to certain methods are also entered in this step. Once the cell is executed, the notebook saves a pdf document detailing the parameter choices and datasets used in the training session.

   4a. If the user decides to use augmentation on their dataset, this is done after the training parameter inputs. Here, the user can use interactive elements in the GUI to perform different augmentations, depending on the datatype the network uses.

4b. If the user wants to train from a previous checkpoint, this is done in an additional step by entering the path to a pretrained model folder. In fnet, models can currently only be trained on the same dataset, and the user will need to provide the number of steps to be added for re-training.

5. Next, the user can initiate the training, by playing the *Start Training* cell. After this step, the model is saved in a folder with the name *model_name* in the *model_path* folder. The training cell also creates a .csv file containing all losses and the learning rate per epoch for most methods.
6. After training, the user can use the Quality Control section. First, the user chooses a model to be used in the section by providing the path to a previously trained model or by using the model used in the same runtime for training. Once chosen, a QC folder is created on the path chosen by the user. In the second cell of the QC section, the user can then play a cell which will display the loss curves during training and save these as .png files in the QC folder. The user enters the paths to an input and a target folder in the final cell. Once executed, the cell will calculate the metrics between targets and the model's predictions on the inputs and provides the quality metrics as an output of the cell for one example, and for the whole dataset in a .csv file which is saved in the QC folder. For the image-to-image translation methods, the cell also creates error maps which are saved together with the predictions from the QC inputs in the QC folder.
7. If the performance of the model meets the user's expectations, it can be used to process datasets for which the user has no ground truth targets and which can be the 'real' data the user wants to analyse using DL. Here, the user inputs the model to be trained, folder containing the dataset to be analysed, and a path where the predicted files should be saved. After playing the cell, these files are saved, and an example of input and prediction is shown to the user in the notebook as a sanity-check.

All these steps run in all the notebooks in code-free GUI. However, all the code remains accessible should errors appear or if more experienced users need to inspect or adapt the code. Together, these steps create the ZeroCostDL4Mic pipeline. The steps we have implemented, and the files saved for the user are designed to maximise the information gained from the training of the neural networks (Fig. 30). While we have refrained from changing source-code for the networks and the training, where possible, the additional features and files created arguably enrich many of the existing tools, if not in their frontend, then at least in terms of information content for downstream analysis or reporting. The table shown in Fig. 30 illustrates this, which compares the output files created in each ZeroCostDL4Mic notebook compared to the original code available from the original repositories.

Method	**Outputs of original code**	**Outputs of ZeroCostDL4Mic implementation**
CARE 2D/3D	Models, TensorFlow.events, predicted images	Models, TensorFlow.events, predicted images, training parameter pdf, csv with training losses and learning rate per epoch, QC csv with SSIM, NRMSE, PSNR scores on test dataset, QC summary pdf, predicted test dataset images
Noise2Void 2D/3D	Models, TensorFlow.events, predicted images	Models, TensorFlow.events, predicted images, training parameter pdf, csv with training losses and learning rate per epoch, QC csv with SSIM, NRMSE, PSNR scores on test dataset (if available), QC summary pdf, predicted test dataset images, predicted images
StarDist 2D/3D	Models, TensorFlow.events, predicted images	Models, TensorFlow.events, predicted masks, training parameter pdf, csv with training losses and learning rate per epoch, QC csv with IoU, confusion matrix values and F1 scores on test dataset, predicted test dataset masks and overlay images, tracking file to use in Trackmate (Fiji)
U-Net 2D/3D	Trained models, predicted images	Models, TensorFlow.events, predicted masks, training parameter pdf, csv with training losses and learning rate per epoch, QC csv with IoU scores, predicted test dataset masks, and overlay images.
YOLOv2	Trained model (best val loss), predicted images with bounding boxes	Models (best val loss, best mAP and last weights), predicted images with bounding boxes, predicted bounding box coordinates (to plot in ImageJ) csv, csv with training losses and mAP per epoch, QC csv with mAP, F1 and confusion matrix values on test dataset, predicted test dataset images with bounding boxes.
Deep-STORM	Trained model, predicted images	Models, TensorFlow.events, predicted images, training parameter pdf, csv with training losses and learning rate per epoch, QC csv with SSIM, NRMSE, PSNR scores on test dataset, QC summary pdf, predicted test dataset images
Fnet	Trained model, predicted images, loss csv files	Models, predicted images, training parameter pdf, csv with training losses, QC csv with SSIM, NRMSE, PSNR scores on test dataset, QC summary pdf, predicted test dataset images
pix2pix	Trained model, predicted images	Models, predicted images, training parameter pdf, csv with training losses, QC csv with SSIM scores on test dataset, QC summary pdf, predicted test dataset images
cycleGAN	Trained model, predicted images	Models, predicted images, training parameter pdf, csv with training losses, QC csv with SSIM scores on test dataset, QC summary pdf, predicted test dataset images

Figure 30: *Summary of data outputs in raw code vs ZeroCostDL4Mic*

II. 2. 4. Characterising Limitations of the platform

While Colab is a platform suitable to mitigate the financial constraint of GPU training, the usage also has constraints that limit the use-cases for the DL tools implemented on the platform. To create transparency about the platform's limitations, we characterised several of the bottlenecks of the platform which affect the type of data or models users can train on the platform.

We identified four areas in which the Colab platform may limit the Deep Learning methods implemented within the ZeroCostDL4Mic resource which we created here.

1. RAM limit – The GPUs provided by Colab have a limited RAM capacity which determines the maximum amount of data which can be loaded into DL models at one time. This is particularly important when training data is usually processed in batches containing multiple images. To estimate the RAM limits, I determined breaking points of the notebooks for all the implemented methods by increasing the batch sizes for a given image size and training session until the notebooks crashed. The results of this test are presented in the table in Fig. 31A and the breaking point batch size (in total pixels in Fig. 31B). To estimate a breaking point, the limit was taken as the number of pixels per batch, which can be calculated as the batch size and image dimensions. Since the way the models handle the training data and the training datasets themselves may vary, these results should be used as guidelines for the maximum RAM capacity of each notebook rather than the absolute limits of the Colab platform. However, these values show that an upper limit of batch pixels lies in the range of between 1e6 to 1e7 pixels except for the Deep-STORM and CycleGAN networks where the limit is reached earlier, likely because the size of the model itself is significant in these models, contributing to the usage of RAM. For example, an image from the segmentation dataset from the isbi challenge 2012 was used in the original U-Net paper and in this project for the U-Net notebook. This limit is equivalent to a batch size of 43 with image or patch sizes of 256x256 pixels. It is not clear how the image's file size relates to this limit, as the size of a single image from the StarDist dataset is almost 10x the size of a U-Net image, yet the StarDist network handles almost 10 times the number of pixels per batch. However, in most cases, the RAM limitation will restrict the size of the patches loaded into the model to the range of 512x512 to 1024x1024, depending on the notebook. However, this does not mean that larger images cannot be used in the notebook, since these can be fed into the

notebooks in patches. In these cases, the image size can be much larger than the maximum patch size.

The RAM limitation does not limit the dataset size used within the notebook per se (see disk space below). However, given the limited number of images that can be loaded into the network per batch, it imposes a constraint in time spent training the models. Since larger batch sizes allow the network to be trained more efficiently, smaller ones may correspond to a higher number of steps per training epoch and thus a prolonged training. This means that users may be more likely to reach the runtime limit imposed by Colab notebooks.

2. Run time limit – The Colab platform imposes a runtime limitation per GPU of 12h. In practice, sessions may also end earlier. This means that training models over days or weeks, which is the case for some of the highest performing DL models in the literature, is not feasible in Colab. However, the limitation can be mitigated by training models across multiple runtimes by using model checkpoints, saved during training. If the models have not converged by the end of a runtime, a model can then be further trained in a consecutive runtime.

3. Disk space – Disk space of the Colab notebooks becomes limiting in two cases. The first case that most users of google drive will encounter is the limitation to data that can be stored within a user's google drive, which is 15GB. This limit can be overcome by either purchasing additional space on Google Drive or by storing datasets in a folder which is shared with the user, e.g., via a lab Google Drive account. The second case is when the session's memory reaches its limit, which is ca. 60GB. This limit can be reached when the dataset used to train the model is very large and is stored in the notebook's session's memory. In some methods, this occurs when models are initialised (fnet), but it more often occurs if datasets, for instance upon augmentation in the ZeroCostDL4Mic notebooks, are saved in the session memory. In this case, the notebook may disconnect, and training will not be possible. In the cases tested in this project, especially for 2D images, the disk space limit is not usually reached and likely becomes increasingly relevant as datasets reach 1000s to 10000s images. For the use-cases in this project, the disk space limitation was not usually the main limitation, as few of the 2D datasets approached this limit. However, for some of the 3D datasets, this limit appears to be reached more frequently and will limit the datasets' sizes that can be used in the notebooks.

4. – GPU access - The final limitation of the Colab platform, particularly when compared with purchased GPU time or local GPU workstations is the reliability of GPU access. Colab provides access to different types of GPU, including Tesla P100, Tesla P1, Tesla K80, Tesla T4, and Tesla T1 GPUs, which are allocated based on demand to the platform, geography, and time of day. (The GPUs I was allocated during a 5-month period are shown in Fig. 31C). Furthermore, models can be trained on only one GPU per runtime. This limitation may be of concern for very large data requirements, in the 100s GB to TB range, which may thus not be feasible to train through this platform. The GPU type influences the speed of training and we have observed an increased speed when allocated to the P100 GPUs compared to the K80 and T4, Fig. 31A. However, performance of the trained models is not expected to be influenced by the allocation of certain GPUs.

In some cases, Colab notebooks will not connect to a GPU and will revert to runtime without acceleration. This can occur when the runtime limit of up to 12 hours is reached. Colab may also 'blacklist' users for a short period of time, usually after using the platform over consecutive days. This 'blacklisting' is often lifted after one day of no use by the respective Google account. This likely ensures a fair allocation of resources if too many users attempt to connect to a GPU runtime.

The limitations outlined above give the boundary conditions for using the Google Colab platform we chose to use for this project and may make the platform unsuitable for certain datasets. However, despite these conditions, a resource based on Colab can still hold value to test models, tune parameters, and explore the merit of DL technology, even if users may ultimately require more powerful resources. Furthermore, in the course of this work, we showed that the limitations do not preclude the platform from being used for relevant real-world biological datasets.

A

Method	patch dim.	bit-depth	# patches	max. batch size	patch dim. x batch size (pixels) at breaking point	GPU	OOM error?	notebook crashes if datasets too large?	out of disk space
CARE 2D	512x512	32	550	21	5767168	K80	yes	no	no
CARE 3D	512x512x4	32	440	8	9437184	K80	yes	no	no
N2V 2D	256x256	32	2816	212	13959168	K80	yes	no	no
N2V 3D	256x256x4	16	96	22	6029312	P100	yes	yes	no
Unet 2D	256x256	8	112	42	2818048	T4	yes	no	no
Unet 3D	256x256x8	8	64?	4	2621440	T4	yes	no	no
StarDist 2D	1024x1024	16	45	10	11534336	T4	yes	no	no
StarDist 3D	128x128x32	16	27	13	7340032	T4	no	yes	no
Label-free Prediction	64x64x32	8	83	16	2228224	P4	yes	no	yes
pix2pix	512x512	RGB	1764	32	8650752	T4	yes	no	no
cycleGAN	512x512	RGB	164	1	327680	T4	yes	no	no
YOLOv2	512x512	8	763	36	9699328	P100	yes	no	no
Deep-STORM	26x26	32	3000	179	121680	T4	yes	yes	no

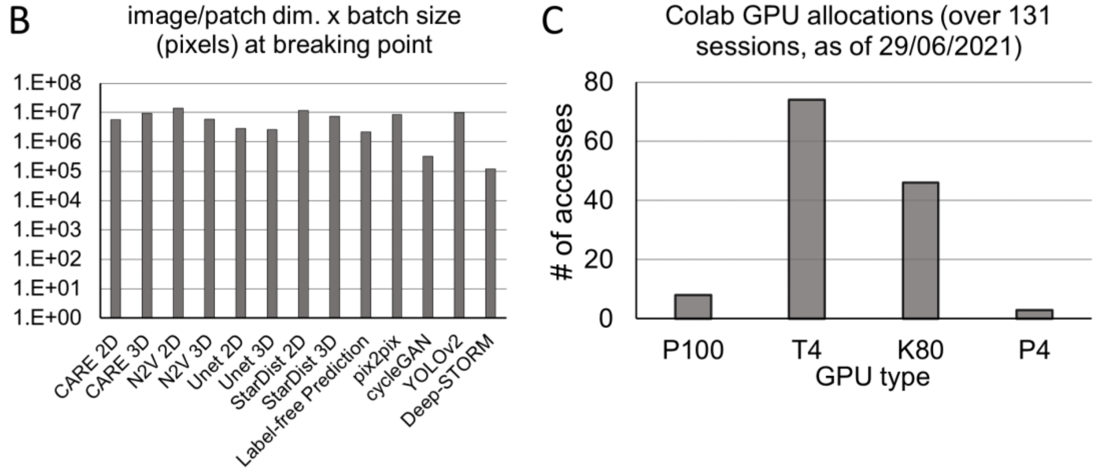


Figure 31 - Limiting factors of ZeroCostDL4Mic in the Colab environment* – A)** *Table shows the parameters used to make the respective ZeroCostDL4Mic notebooks crash and the errors to be expected.* ***B) *Histogram of the maximal batch size for each notebook, this does not appear dependent on GPU type in Colab.* ***C)*** *Histogram of the allocation of GPUs per training session.*

II. 3. Discussion

In this project, we created the ZeroCostDL4Mic platform which aims to address the main barriers that currently prevent a larger proportion of the bioimaging community to use DL-methods gainfully and successfully for their analysis tasks. Although ZeroCostDL4Mic is characterised in detail in its publication, in this chapter, I showed what challenges were faced in constructing the platform and, specifically, how the overall consistency of the notebooks could be maintained while the underlying methods and their code implementation by developers differed significantly. In these details, it is important to convey with more clarity the challenges that users of DL-tools might experience when adapting published DL-tools to their own bioimage analysis problems. Hence, the key result of this chapter is not only the end-product, the ZeroCostDL4Mic platform, but the experience gained from creating it. The first conclusion drawn from this experience is how many more challenges need to be overcome to implement certain tools into the platform than others. Here, I covered four of the core methods of ZeroCostDL4Mic, which broadly represent the types of methods used throughout the project: CARE is a representative for the methods developed primarily by a group in Dresden who also created the StarDist, Noise2Void and DenoiSeg methods which are also part of the platform, U-Net 2D is an example of a Keras method which is entirely contained within the Colab notebook, of which U-Net 3D and Deep-STORM are further examples in the platform. YOLOv2 represents a method which uses Keras but depends on external scripts to be implemented. At the same time, fnet 2D/3D is PyTorch based, like pix2pix and cycleGAN, and requires several external files which are accessed to train a network.

The methods developed by the CSBDeep group stand out on this list in terms of ease of use and ability to bring DL to the bioimaging community for several reasons. CARE, like the other CSBDeep methods, is constructed as a Python package and well documented. This facilitated understanding of CARE's functions and their use cases and implementation of this method in ZeroCostDL4Mic. Since all the CARE hyperparameters for training and the paths to the training datasets could be used directly as input arguments for the CARE model creation and training functions, it was relatively easy to implement in the GUI in this project. This is also the case for much of the U-Net 2D implementation. An additional attractive feature of using Keras to implement U-Net 2D is the ability to use Keras data generators in training.

This is particularly useful in the Colab environment as it makes efficient use of the environment’s RAM and does not require datasets to be buffered in the runtime which can eat into the total time available before notebooks time out.

Compared to CARE and U-Net, the YOLOv2 and fnet methods were difficult to implement in the notebooks. The latter methods are designed to use script files to store paths and parameters for training and to execute certain functions, including training and prediction. This is not necessarily a flaw of these methods and is a valid design choice by the developers, likely to allow users to see all their parameter choices in one place instead of searching and changing them inside a code-block. However, neither of these methods was as well documented as the CSBDeep methods, and in their raw form would be challenging to use for users with little experience in coding. The lack of annotation and the structure of the repositories, with many functions importing methods from other scripts in the repository, made integration into the ZeroCostDL4Mic format challenging.

The key to using these methods in the notebooks was to make changes in these script files by editing their choices. This was primarily done using shell commands which can be easily used in the notebooks via *!* at the start of the line. However, using these commands can be error-prone and is difficult to debug as failure to execute commands does not always result in an error message and failure to insert a specific parameter usually only becomes apparent by opening the file and verifying that the parameters were correctly inserted or when error messages appear further downstream in the code. Secondly, any parameter search needed to be unique, which means that the search term could only appear once in each script file, as otherwise any edits would appear in multiple locations in the document. This was simpler when only individual parameters needed to be inserted into script files (Fig. 14 and 15).

It was more difficult to insert new functions or parameters into scripts to be able to fulfil some of the aims designed for this project. Here, entire new lines of code needed to be inserted. An example of this in fnet was evaluating a validation loss during training, which surprisingly is not done by default in fnet (Fig. 15C). In YOLOv2, a similar change needed to be made in the YOLOv2 frontend.py to ensure that mAP was evaluated as a callback (not shown). These types of changes would be relatively simple to make if script-writing was expected of the user, but because this project specifically set out to avoid user-side code-input, these relatively complex steps needed to be performed in the notebook, upon playing the cell.

Comparing the different methods in terms of their ease of implementation gives insight into their general ease-of-use and their applicability for a wide variety of tasks. Looking at the above observations, it can be concluded that the methods requiring the least effort in terms of their application are tools which do not use multiple scripting files and instead design their tools for user-friendliness such as the CARE methods or by being able to use Keras entirely. Although there is a rationale, e.g. to store all hyperparameters in a separate document, which can also be of use for the user after the network is trained, inputting all parameters in a script file is usually no more convenient than inputting these parameters in a Python object, such as a configuration library, as is done in both CARE and U-Net, thus, reducing the number of necessary documents to run a method. In ZeroCostDL4Mic, we ensured that the most important hyperparameters, essentially those chosen by the user in the notebook, are stored for every model in a human-readable pdf format, such that even for methods that do not by default store parameters in a configuration file, they remain useable for review after training, such as for U-Net and CARE. This pdf serves an important function for users who want to learn about the effects of models, datasets, or parameter choices on the performance on a specific task, specifically when returning to models which were trained a long time ago.

Despite the differences and the ensuing difficulties in implementing methods such as YOLOv2 and fnet, the input cells for paths and training parameters could be kept extremely consistent across notebooks which means that users should recognize the similarities between methods despite the underlying differences. The key differences that do exist, e.g. choosing the number of layers in U-Net, or the penalties in YOLOv2, are specifically adapted to each method and give the user opportunities to experiment and optimise their methods as much as possible. In some cases, it would have been possible to add further parameters. For example, it has been observed that changing the size of the convolution filters in U-Net-architectures can impact performance. However, this may have required further adjustments, either within the methods themselves, or would likely unnecessarily complicate the methods for the intended user base. However, the included parameters that users can choose from should be sufficient to tune models to sufficiently high performance, something I explore in chapter 2.

Implementing methods for predicting unseen data or for the quality control step involved similar strategies for adapting methods to the ZeroCostDL4Mic environment. Here, more than inputting any user-specified parameters into Python objects or script files, the challenge was to save all predicted files in a similar and accessible format. Again, this was more challenging for fnet and YOLOv2 than for the CARE and U-Net methods. While the latter save files automatically in a path that can be input to the function which executes the prediction, in fnet several changes needed to be made to the prediction script (predict.sh or predict_2d.sh) which included removing the option of predicting on training data or including targets for the prediction. These choices by the developers were difficult to comprehend when developing the notebook as they would likely make this method difficult to use for any users whose data is not exactly formatted as intended by developers. A further inconvenience in the default fnet prediction pipeline was that any predictions were saved in folders named with consecutive numerals so that the original file name was no longer visible, with each prediction saved in a folder together with an input and a target. However, it was useful to save information on the filenames and even more so in Colab without requiring the input in the same folder, since memory is limited in Google Drive and Colab, and saving each input in the results folder is not memory efficient since the user already has access to their own input files. The choices made in the development of fnet were often counter-intuitive and suggest that fnet was developed for a very specific dataset or dataset-type without considering the challenges in adapting it to different use-cases. The efforts made in this project to simplify the DL-experience may, therefore, improve the experience of users wanting to use this method, potentially growing its user-base.

In YOLOv2, a particular problem in the prediction function was that the predictions were only saved as images with bounding boxes directly drawn on the images. This makes downstream analysis extremely difficult. Interestingly, this is not a peculiarity of YOLOv2 and indeed, many object detection methods create only images with bounding boxes as prediction outputs, despite also calculating coordinates for each box, which would arguably be a more valuable output for downstream analysis. This means that the output of the neural network model is in a different format to its training input, which usually consists of annotation files, in the case of the ZeroCostDL4Mic implementation of YOLOv2, in the.xml format used for the PASCAL VOC detection challenge. Creating image outputs with bounding boxes instead of a set of coordinates may be related to the types of tasks tools such as YOLOv2 were originally designed for: the detection of real-world objects on public image datasets. However, for bioimaging, object detection methods should output much more quantifiable information, ideally in a format which can be used quickly for downstream analysis. In ZeroCostDL4Mic, the prediction function of YOLOv2 was therefore changed such that the predicted bounding boxes were saved as coordinates in a .csv file. This .csv file is firstly readable by a human and can secondly be imported into Fiji, to view and edit any predicted bounding boxes which would not be possible using only the default outputs of YOLOv2.

One feature in the prediction functions that re-occurred was that many prediction functions take only individual images as inputs, which required these functions to be implemented in a loop in the notebooks to use a folder as an input. This was somewhat surprising since developers should presume that users will employ their methods on batches of data.

Again, in this project, we largely succeeded in making the prediction section and its outputs very consistent across DL-methods, despite the underlying differences and in some cases, arguably made improvements in terms of data-structure. This should help users make use of these methods more efficiently after training is complete than is the case with the default methods.

When building the ZeroCostDL4Mic platform, one of the most important novelties integrated into the workflow was the addition of the QC step which is rarely integrated into DL-methods, even though they are essential for reporting the quality of the performance of a method. In the ZeroCostDL4Mic notebooks, the quality metrics and error maps were calculated by performing the calculations directly in the notebook (NRMSE, IoU), by using Python packages (SSIM) or by importing and integrating Python scripts from public repositories (mAP). However, none of these metrics were already integrated into the methods themselves when the notebooks were created. This is an indication that making robust quality controls on trained models accessible to users has not been the focus of developer groups. However, the scepticism levelled at DL methods in bioimaging is often directly concerned with the lack of transparency of DL outputs. Here, the integration of QC into the workflow is a direct response to the existing shortcomings of methods currently developed in the community and will hopefully be helpful in building trust in these methods.

In addition to our core workflow, we included two additional steps which should make DL more accessible for our notebooks. The first was the integration of data augmentation. Users would likely perform this step before training a model, but it was included in the workflow as it should, ideally be a very simple part of the DL-workflow and be effortless and intuitive. Augmentation is likely the least consistent step across the notebooks because different methods were convenient for each data type and because the underlying augmentation functions or packages performed augmentations differently. Here, Keras augmentation was the easiest to implement because it only involved inserting user choices into a Keras object which carries out augmentation dynamically during training. Here, the augmentations are performed on a percentual basis where the user determines how likely a certain augmentation should occur. Although this form of augmentation is convenient in terms of implementation, I believe it is not the most intuitive form for users. It is usually not clear how much the dataset is augmented and what the input images look like. In U-Net, we attempted to alleviate this small problem by displaying examples for augmentation after the augmentation cell, which we also did for all the other notebooks. However, it is still more intuitive to give the user the option to choose an augmentation factor implemented in most notebooks although the underlying methods for augmentation differ. Here, the disadvantages are primarily the lack of memory to save augmented images in the runtime (which is not a problem when using a datagenerator) which can be slow and consumes resources the user may want to use for other purposes. Hence, the augmentation sections often compromise intuition and their convenience of implementation. Importantly, all notebooks have an augmentation option and can be used with interactive tools in the runtime. Such visual features are an additional element which should give the impression that DL methods can be used to experiment or even play with in the notebooks. Despite the relative inconsistency between different notebooks, the augmentation section always remains simple to apply in line with the platform's design principles.

In concluding this chapter, I will summarise the key features that I found important for the practical use of a DL method. In this chapter, I was primarily concerned with the question how an intuitive GUI can be built from the existing code-resources available for the bioimaging community. In this process, I have made several observations on how developers currently implement code and can outline key features which I believe would make any future methods more accessible than many current tools in the community. These are the same aspects I tried to integrate in this platform. These recommendations follow the workflow of the suggested DL pipeline.

Documentation

1. Tools must be well documented, specifically, the meaning and purpose of parameters used for training.
2. Developers should provide example code and importantly, example datasets (including meta-data) that are easily accessible to users. Especially, the availability of an accessible example dataset may facilitate reproducibility of results by users on their own datasets.

Installation

3. Tools should be easily downloadable by the user with few steps, e.g., a Python package via *pip install.* Users should not be required to download or install other software than the one of the specific DL-package.

Training

4. Training the model should be simple and not require additional files to store code or paths.
5. Validation during training should be performed by default. Users should be able to opt-out but should never be required to opt-in to performing validation during training.
6. DL-methods should automatically output as much information as possible about hyperparameters and training history in a human-readable format, so that experiments can be analysed and replicated easily by the user themselves or reviewers.

Quality Control

7. Quality Control methods should be integrated by default in every DL-tool.

Prediction

8. Outputs should be saved with the original filename still available and with as few additional folders as possible.

9. Outputs should be in a format that allows downstream analysis and should be compatible with other tools used by the community.

Data types

10. Tools should be able to handle different data formats and bit-depths as bioimaging data comes in various formats. It is cumbersome for users to adapt their data to the specific needs of a method. The method should adapt to the data, where possible.

Efficiency

11. Methods should be designed for memory efficiency. To exploit large datasets on available hardware, methods should be designed and tested to minimise, e.g., the RAM needed for training. This makes the methods faster and, therefore, more likely to be used by analysts.

I believe that these recommendations beyond being guidance for good practice in development of new tools, also provide a good summary of what I attempted to achieve when creating this platform. While hardware access is often a significant barrier for DL, the lack of relatively simple features in code should not be underestimated as an obstacle to the democratisation of this technology. The development of languages such as TensorFlow, Python and Keras has been a significant driver in this field and has already helped its democratisation among coders as it has dramatically simplified building and training neural networks. The next step which should bring this technology to an audience beyond coders is its adaptation to a code-free environment which requires good practices and intuitive and useful design elements. With the implementation of the presented methods, together with my collaborators, I took the first steps in this direction.

II. 5. Methods

Testing the notebooks

During the development of the notebooks, the workflow was tested by a group of beta-testers with little to no previous experience with DL. Their feedback helped to design the final workflow and layout of the notebooks.

Notebooks were tested before release by me, Romain Laine, and Guillaume Jacquemet to ensure they function as intended in the following way. Each notebook is played cell by cell with transfer learning and augmentation disabled or skipped and tested on the publicly available datasets (zenodo) created for each method[197–203]. Once it is fully run, the runtime is disconnected, and a new runtime is initialised. The cells are run as before, the dataset is augmented, and the previously trained model is loaded into the transfer learning cell. The other cells are run as before. The runtime is disconnected a second time, and the notebook reconnected. The training section is skipped and one of the previously trained models is assessed in the QC section by disabling the *‘Use_current_trained_model’* tickbox and entering the path of the previously trained model in the *‘QC_model_path’.* Once QC is complete, the runtime can be disconnected again and re-established as before. In the final test, the training and QC steps are skipped and analogous to the previous step, the tickbox *‘Use_current_trained_model’* is disabled in the Prediction cell and the path to one of the previously trained models is entered in the *Prediction_model_path*. The prediction cell is played. If all these steps run error-free on a respective dataset, all input cells apart from those containing default parameters are cleared. The outputs of all cells are cleared, and the runtime is disconnected. The notebook is saved and ready to be released in the repository.

Datasets

CARE 2D

Guillaume Jacquemet acquired the dataset shown in Fig. 12 (CARE 2D) for the ZeroCostDL4Mic project[134,197]. Briefly, images of fixed DCIS.COM LifeAct-RFP cells were acquired on a DeltaVision OMX v4 SIM set-up with a 60x Plan-Apochromat objective lens in SIM mode for high resolution images of actin labelled with Phalloidin-488 at high laser power (10%) with 50ms exposure. Low SNR images were acquired by imaging the LifeAct-RFP channel with lower laser power (1%) with 100ms exposure. For more details see details in [134].

The data shown in Fig. 21 was acquired from a fixed sample of HeLa cells with EGFP α-tubulin on a Nikon ECLIPSE Ti-E with a 60x Apo VC oil immersion objective, in widefield mode. The difference in noise between inputs and targets was achieved by different exposure times with low and high noise achieved with 500ms and 5ms exposure, respectively at an excitation wavelength of 550nm.

U-Net 2D

The data used for U-Net, as shown in Fig. 13 and 22, is from a public dataset which was used in the original isbi segmentation challenge in 2012[112].

YOLOv2

Guillaume Jacquemet acquired the raw data shown in Fig. 14 (YOLOv2) for the ZeroCostDL4Mic project[134,198]. Briefly, breast cancer cells (MDA-MB-231) migrating over a cell-derived matrix were imaged on an inverted wide-field microscope (AxioCam MRm camera, EL Plan-Neofluar 20/0.5 NA objective (Carl Zeiss)). The data shown in Fig.17 shows HeLa cells (HTRG) with endogenous nuclear GFP expression imaged in a Nikon ECLIPSE Ti-E microscope under widefield settings using a Plan Apo 20x oil immersion objective and excitation wavelength of 525nm in a timelapse with 1-minute timeframes and 40ms exposure. Cells were previously grown to confluence at 37 degrees and treated with a cycle inhibitor for 12 hours prior to imaging, which was washed off the sample before timelapse acquisition by gently aspirating the growth medium (Optimem with 10% FBS) and pipetting fresh medium into the plate and repeating this step 3 times. The annotations for the YOLOv2 datasets were performed using the public makesens.ai platform (https://www.makesense.ai/). This is described in detail in [134]. Briefly, the images were imported into the makesense.ai platform, manually annotated with class-labels and bounding boxes. The annotations were then downloaded as .xml files in the PASCAL VOC format for use in the ZeroCostDL4Mic notebook.

fnet

The data shown in Fig. 15 for fnet shows HeLa cells with labelled with H2B mCherry red which were grown to ca. 70% confluence at 37 degrees and 4% CO2 in Optimem medium (10% FBS). The two channels were acquired using a LEICA SP8 confocal microscope with a HC PL APO CS2 63x oil immersion objective. To image multiple channels simultaneously, the brightfield channel was acquired with a transmitted light detector and the fluorescent channel with a photomultiplier tube (PMT). Timelapses were acquired excitation wavelength for the fluorescent channel was 570nm and exposure time 30ms, with 1-minute timepoints.

Software versions

All the notebooks use Python 3.2. All notebooks use TensorFlow 1.15.2 as the default backend, with the following exceptions: N2V 2D uses TensorFlow 2.1, fnet uses PyTorch v 1.10 and CUDA 11.1., CycleGAN and pix2pix use PyTorch v1.19 and CUDA 11.1.

Data availability

All the notebooks in this project are publicly available on the ZeroCostDL4Mic's GitHub repository https://github.com/HenriquesLab/ZeroCostDL4Mic and all datasets used for testing the notebooks are available on the project's Zenodo repository accessed through GitHub or the publication to this project[134].

III. Results Chapter 2 – Quality Control (QC) of DL-tools in ZeroCostDL4Mic

III. 1. The goal of QC in ZeroCostDL4Mic

In the previous chapter, I outlined how the need for accessible DL for bioimaging tasks inspired the creation of the ZeroCostDL4Mic platform. The primary goal of this project was to facilitate training DL models on custom datasets. As indicated in the Introduction to this work, a significant aspect of disseminating DL technology is ensuring that the available tools produce meaningful and reliable output (Reliability Problem). To ensure that the models created through the ZeroCostDL4Mic pipeline can accomplish this, it needed to provide quality control methods (shortened: QC). The QC step is not always considered part of the DL pipeline. Many available tools provide quality metrics only in the form of additional optional functions for the users, such as CARE[113], YOLOv2[94] and Label-free prediction[114], and generally require additional effort from the user side to implement.

As explained in the previous chapter, in ZeroCostDL4Mic, QC is an integral step of the workflow between the training and the inference step. How does the integration of this step aid the democratisation of DL tools? There are at least two answers to this question. Firstly, QC can show that the models trained on the platform can perform the tasks they are designed to do. This means that the QC section fulfils the purpose of confirming in a quantitative, thus comparable, manner that the tools implemented in this project achieve a specific performance. The key task here is to examine if desirable outcomes can be achieved with the dataset size, runtime, and RAM limitations present in the Colab platform. If not, the QC step will at least show any shortcomings and can thus help users to determine alternative routes of investigation than Colab or even DL-based tools. The second way an integrated QC step allows users to adopt DL strategies is that it can be used to optimise DL-models, by performing the training step using different parameters and then evaluating how such changes affect performance. With the guidance provided in the notebooks, this could help in creating a higher degree of literacy with DL tools, and how fine-tuning of models can be achieved, and potential problems overcome, thus targeting the knowledge problem (see Introduction).

For these two aims to be fulfilled, the QC section needs to accommodate the following key features, which will be the subject of this chapter. First, the metrics used to evaluate the models on the platform should have some use in the community for the relevant tasks. Second, their predictions should provide a degree of interpretability of the performance of the models. However, beyond the metrics, the QC step should provide a qualitative assessment which means that it should be possible for the user to corroborate any quantitative metrics themselves. This means that QC can be performed in a quantifiable way that makes trained models comparable with each other. The performance of the models is always transparent to the user, which may not be the case for purely quantitative metrics.

For the second feature required of the QC section, the ability to use it to fine-tune parameters for training, it should be shown that any metrics, quantitative or qualitative, are sensitive enough to detect performance changes in the trained models for changes in the training datasets and hyperparameter settings. In this chapter, I show that the QC section achieves this, representing an essential contribution to allowing DL to become democratised and easier to use within the bioimaging community. To demonstrate this, I focus on the main tasks performed by the implemented tools in this project, object detection, object segmentation, denoising, and artificial labelling. To avoid redundancies, I will focus on one tool for each task, covering the quality metric generally used to assess these tasks. This provides an overview of how QC can be used in this project to improve the reliability of DL tools in microscopy and bioimage analysis. It is important to note that it was not possible in this project's scope to perform an exhaustive optimisation of all hyperparameters for all included DL tools. Instead, I have focused on specific parameters and settings in the notebooks considering the type of dataset, the size of the dataset, and the time required to train the models in Colab. Hence, the sections, divided by task, will cover different strategies to optimise model performance. This aids in understanding which choices users have in the notebooks without repeating the same steps in each section. For example, while augmentation was shown to improve performance in this chapter only for YOLOv2 and StarDist, it can also be used for the other methods, although it is not shown here. For instance, it would have taken a significantly longer time, in the order of multiple days, to train fnet even a single time if the dataset used here had been augmented, which would have made it difficult to test with different parameters, such as patch size which was done in this chapter.

In the process of testing the use of the QC section, three key observations were made: Firstly, all the methods trained here can achieve a desirable performance on representative biological datasets from microscopic acquisitions. This is important because it shows that custom research data, rather than public datasets, can be used in ZeroCostDL4Mic. Further, with one exception, all networks can be trained to convergence, the exception being fnet, which shows overfitting without the validation loss converging beyond the first steps. However, this is not reflected in the performance of the models in unseen data sets where the fnet models trained for longer performed better according to quality metrics.

Second, all the tested models can be improved with parameters and settings available in the notebooks, making the QC suitable for parameter optimisation experiments, an important difference between this tool and models available in model zoos.

Finally, for the case of metrics used for the denoising of three biological datasets, I find that the currently widely used SSIM, MSE and PSNR metrics have shortcomings when used to evaluate model performance. When used without consulting an error map or an image, these metrics seem biased towards images with more background regions, which can skew the analysis of model performance on different datasets when this is not controlled, which currently is not common in publication[113,114]. This observation revealed an important benefit of the QC section in this project: the availability of error maps for the squared error and structural similarity metrics, which are shown and saved for the user. These error maps provide transparency to model performance which is not given by the metrics alone. They resolve image areas that are not well predicted by the model and reveal if the model learns mostly background information.

These results suggest that current metrics for the analysis of models performing any form of image reconstruction do not sufficiently reflect model performance unless supplemented by visual inspection by a human.

III. 2. QC for an object detection task in ZeroCostDL4Mic

Identifying the objects of interest in images is a common task in bioimage analysis and one in which DL-methods have shown significant superiority compared to classic algorithms. In ZeroCostDL4Mic, the task of classification and object detection was first implemented with the YOLOv2 method[94] (see Results chapter 1). Evaluating the performance of object detection algorithms can be achieved with intuitive metrics which underly other higher-level metrics: true positives (TP), false positives (FP) and false negatives (FN). These are evaluated against the total number of objects labelled in a test dataset. As explained in the Introduction, these metrics can be combined to give even deeper insight into the performance of a model by evaluating recall and precision metrics (see Results Chapter 1), which can be combined to create the F1 score or to create the so-called p-r curves to visualise the performance graphically. Together these metrics provide a ready quantification of model performance on a dataset that can be easily compared between models.

Several models were trained to classify and detect cells of different morphologies in a small image dataset to investigate whether the notebook implementation of YOLOv2 could be used in a standard object detection task. The initial question for this dataset was if accurate detections can be achieved in the ZeroCostDL4Mic notebook on a small dataset of 30 images, i.e. a dataset compatible with the limitations of the Colab environment and those relating to the challenge of curating a dataset for a classification task. The chosen dataset contains images of migrating cancer cells (MDA-MB-231) collected by a collaborator of the ZeroCostDL4Mic project[198] (Guillaume Jacquemet) and consists of a total of 30 images for which three classes of cells were distinguished: migrating, rounded, and dividing. Out of this dataset, three images containing instances of all classes were set aside as test images and the remaining 27 images were split into 24 images for training and 3 images for validation. A bigger dataset could have been assembled as more raw images were available. However, labelling all cells was challenging, with up to 80 cells per image and a total of more than 1200 annotations, sometimes with cells of challenging identity, making the annotation of the dataset laborious and time-consuming. This dataset, therefore, represents a challenge for YOLOv2, which was initially tested on thousands of images[94]. However, many datasets by the envisioned user-base will likely have datasets of similar size and complexity. Hence, this project is interested in determining whether the YOLOv2 implementation could even help users in an object detection challenge.

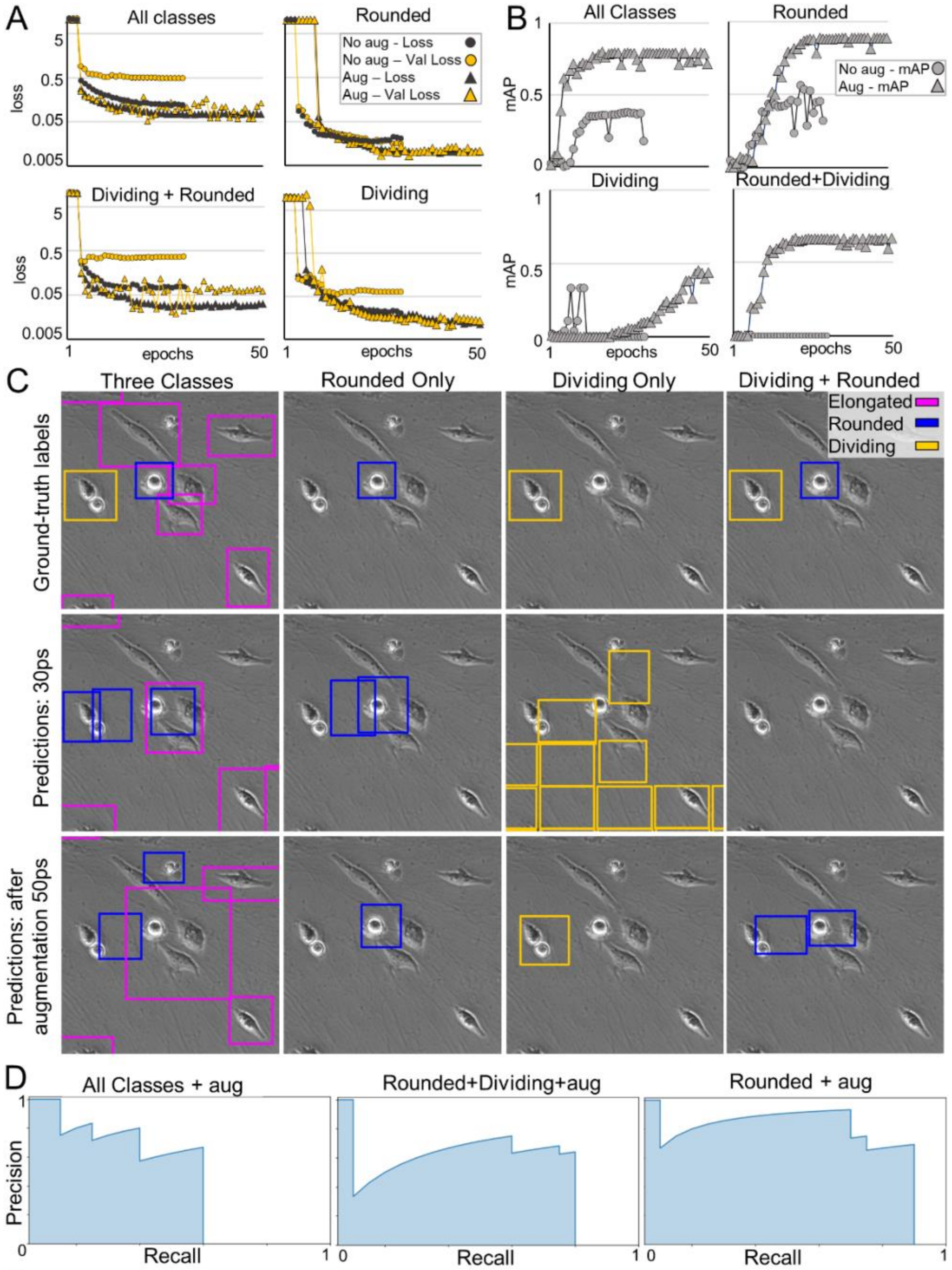
In the first trial, a YOLOv2 model was trained on all classes for 30 epochs, using the default settings of the notebook. This included three warm-up epochs in which only the first network layers are trained. This is suggested as an aid for the network, which uses pretrained weights from a large public image dataset, to adapt to a novel dataset[94]. To test if the datasets were sufficient to train a YOLO model, I first tested if training and validation losses converged. For YOLOv2, the loss measures several terms, bounding box accuracy, bounding box size, classification precision and penalizes the models' confidence in wrong predictions (for details, see [204]). After a warm-up, losses dropped dramatically, suggesting that the neural network learns the classification task (Fig. 32A). However, validation and training losses significantly diverge (Fig. 32A – top left), suggesting the model overfits the dataset. When investigating another training metric, the mean average precision (mAP), which measures the average precision over all classes, the scores remain low (below 0.5), which appears to confirm relatively poor performance on the full dataset (Fig. 32B – top left). Here, the task of model optimisation begins, which will be required to achieve better performance for this task, if possible. As indicated above, demonstrating the ability to improve model performance is essential for the ZeroCostDL4Mic tool to be of use for novice users of DL tasks and research purposes. Here, the most likely problem for the model is the small size of the data set. Hence, the first step to improve performance was to augment the dataset using the augmentation section built into the YOLOv2 notebook. Here, I performed the maximal available augmentation in the notebook, which increased the size of the dataset by a factor of 8, yielding a total of 192 training images and extending the annotated cells in the dataset to over 10.000. The training was repeated with the same standard settings with this augmented dataset. Here, the losses immediately showed an improvement compared to the non-augmented dataset, and there was no apparent overfitting even when training was extended to 50 epochs (Fig. 32A – top left). The mAP in the validation set also improved almost immediately, plateauing at 0.75 (Fig.

32B – top left). Since the larger dataset appeared to solve the problem of overfitting seen before augmentation, further performance evaluation was performed to investigate the models in more detail. Here, it became evident that despite mAP scores that suggested robust classification and detection performance, the model provides only relatively coarse classifications. Although several cells in the FOV, especially in the 'elongated' class, are well classified and detected with bounding boxes, other cells only have coarse bounding boxes or are not correctly classified or detected (Fig. 32C bottom left). An interpretation of this visual inspection is that since the model appears to be worse at detecting the ‘rounded’ and 'dividing' classes, there is a problem in learning these classes from the dataset. Indeed, the problem likely arises from the class imbalance in the dataset as ‘elongated’ cells are nearly five times more common than ‘rounded’ and ‘dividing’ cells. This means that the ‘elongated’ class dominates performance measurement during training. How can such a problem be overcome? An option could have been to reduce the number of ‘elongated’ annotations in the dataset. As this would have required modifications of the annotation files associated with additional effort, I instead deleted this class. A further rationale for this step is that in many classification tasks, the main task consists of identifying specific cells, such as cancer cells or dead cells, in a large population of cells which can be mimicked here by training a model to detect and classify only two ‘rare’ cell types, in this case ‘rounded’ and ‘dividing’ cells.

Therefore, the experiments above were repeated with datasets that did not contain the ‘elongated’ class, removing these labels from the annotation files. Furthermore, I created two datasets that only contained a single class, which resulted in three datasets, one containing ‘dividing’ + ‘rounded’ cells and two datasets containing only ‘rounded’ and ‘dividing’ cells, respectively. Repeating the above experiments with these datasets showed strong training performance in ‘rounded’ cells in the standard and the augmented datasets, yet the mAP scores for the augmented dataset significantly improved the standard dataset, peaking near 0.9, suggesting accurate predictions. The models trained on the dataset containing two classes and the one containing only ‘dividing’ cells showed overfitting similar to the original dataset, which could be compensated for the ‘dividing-only’ dataset by augmentation, but not in the mixed dataset. However, the overfitting behaviour of the latter model appears less significant than in the original dataset, with the validation loss diverging late and remaining constant parallel to the training loss. This may be caused by some particularly ‘challenging’ cells being randomly allocated to the validation set, which the model could not correctly predict. This may have led the model to converge at a local minimum and may have prevented the correct class-label from being learned. This could explain why the validation loss appears to suddenly diverge late during training. However, the mAP values suggest that the model has problems correctly classifying cell morphologies, as the value plateaus are near 0.6. These problems may be caused by the model's difficulty in detecting dividing cells, which is also the class with the lowest frequency in all datasets, with only 200 instances existing even in the augmented dataset. In the first trial, without augmentation, the 'dividing only' trained model does not settle at a mAP greater than 0. Only with augmentation does the mAP score rise over training, albeit slowly, and does not peak at 50 epochs. More extended training, which was not performed in this case, could have helped increase performance slightly.

Comparing the ground-truth with the models’ predictions shows that the models trained on the individual classes did detect the cells that the original model struggled to classify and detect, however only after augmentation of the datasets (Fig. 32C – middle columns). In the mixed dataset, the trained model fails to detect any cells without augmentation but detects two cells when augmentation is applied. However, the bounding boxes do not cover the cells well and the dividing cell is incorrectly classified as ‘rounded’. As in the original dataset, this result is likely caused by the class imbalance between the ‘dividing’ and ‘rounded’ classes. This suggests that for detection tasks in ZeroCostDL4Mic, rare class instances in a small dataset are likely to be best detected by models trained solely for this class.

After visual inspection, quantitative quality control metrics help identify further potential problems. Looking first at the p-r curves of individual classes for all models shows that with the reduction of classes, the performance for individual classes improves. This is indicated for the ‘rounded’ class in the augmented datasets in Fig. 32D. Especially, the final result (right) suggests that the model correctly labels even challenging cells, i.e. cells which are correctly identified for high recall values (see Methods). Further insights can be gained by looking at specific TP, FP, and FN counts, as this provides a dataset-wide evaluation instead of an individual example. The results mostly support what was seen above. All models improve with augmentation and more training epochs, as indicated by the losses and mAP scores in training. Furthermore, the counts of the quality metrics show that the test set contains only 3 cells of the ‘dividing’ class which will lead to large ‘jumps’ in performance when cells are or are not detected in the test set. Yet, two out of three of these presumably difficult-to-detect cells were identified with the augmented ‘dividing-only’ trained model, which suggests that performance even for difficult classes can be achieved if the training dataset is adjusted, e.g. by augmentation. However, it also suggests that the test dataset and the training dataset should contain more balanced classes and probably more training images.



E

Training classes	epochs	Test Classes	FP	TP	FN	recall	precision	f1 score
All classes								
elongated, dividing, rounded	30	elongated	11	50	79	0.388	0.82	0.526
		rounded	10	12	8	0.6	0.546	0.571
		dividing	0	0	3	0	0	0
elongated, dividing, rounded - augx8	50	elongated	22	81	48	0.628	0.786	0.698
		rounded	9	12	8	0.6	0.571	0.585
		dividing	0	0	3	0	0	0
Mixed Classes								
Rounded+dividing	30	rounded	44	0	20	0	0	nan
		dividing	5	0	3	0	0	nan
Rounded+dividing - augx8	50	rounded	12	16	4	0.8	0.571	0.667
		dividing	0	1	2	0.333	1	0.5
Individual Classes								
rounded	30	rounded	5	15	5	0.75	0.75	0.75
rounded - augx8	50	rounded	10	18	2	0.9	0.643	0.75
dividing	30	dividing	188	1	2	0.333	0.005	0.01
dividing - augx8	50	dividing	1	2	1	0.667	0.667	0.667

Figure 32 – YOLOv2 evaluation on a small dataset of migrating cells – A) *YOLO loss*[204] *for models trained on datasets with different class counts.* ***B)*** *mAP scores during training evaluated on the validation dataset for the same models as in A.* ***C)*** *A representative field of view of the test dataset and its ground-truth labels (top) and predictions by models trained on datasets with different class counts, without (middle row) and with dataset augmentation (bottom row).* ***D)*** *p-r curves for the 'rounded' class. The titles above the curves indicate the dataset used during the model's training.* ***E)*** *Table of the quality metrics evaluated on the test dataset for models trained on datasets with different class counts.*

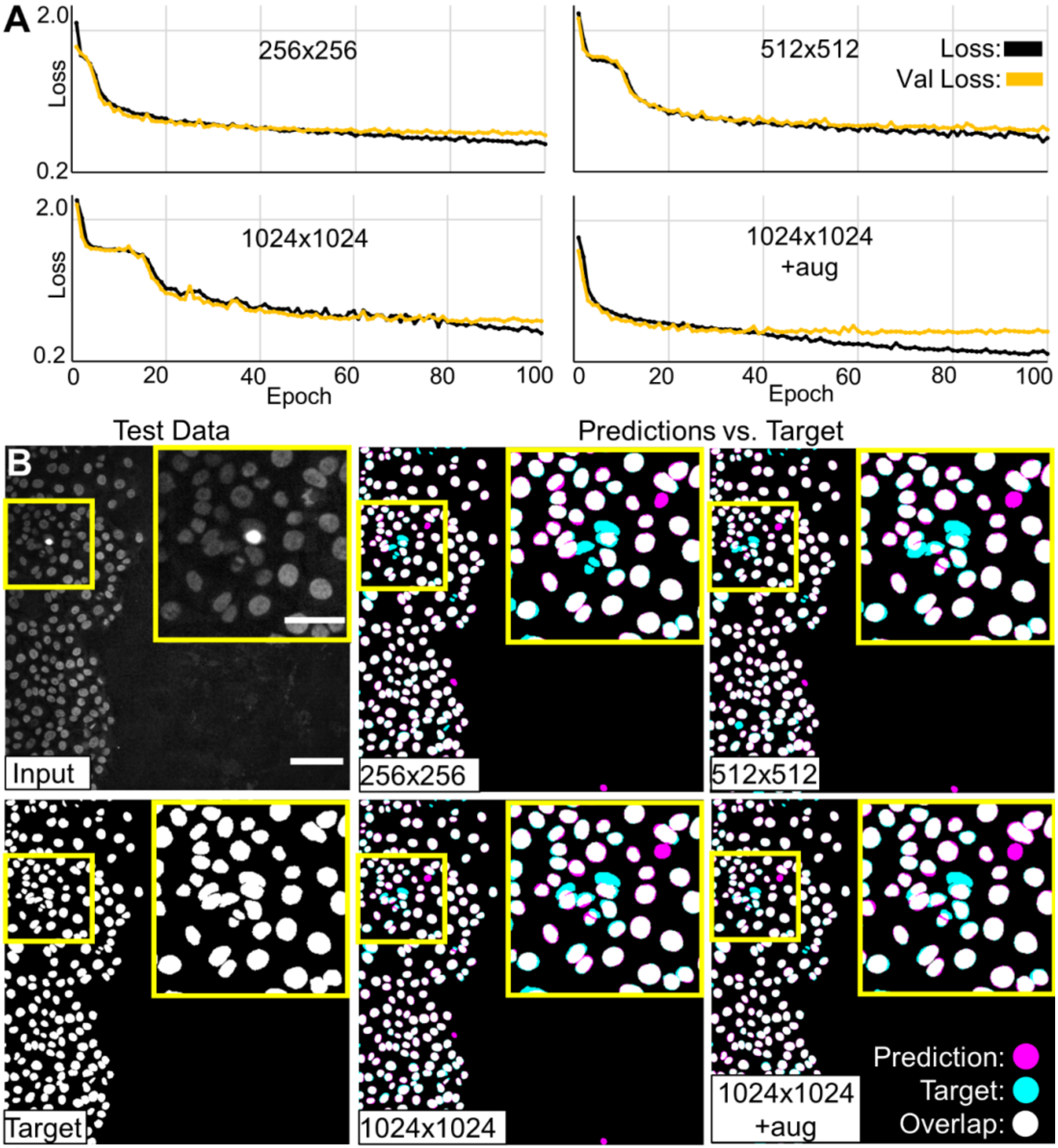
III. 3. QC for a segmentation task in ZeroCostDL4Mic

To test how the QC section could be used to evaluate and improve models for a segmentation task, I chose the StarDist 2D notebook as an example, a method designed for nuclear segmentation. Demonstrating the QC task on StarDist 2D has the advantage that more features can be investigated than in the ZeroCostDL4Mic U-Net notebook, which can only be evaluated on one metric, namely the intersection over union (IoU) score. In StarDist 2D, the QC section contains the IoU and other quality metrics which evaluate the ratio of detected vs. undetected cells and, therefore, covers a more diverse range of metrics than the U-Net notebook. Guillaume Jacquemet primarily created the StarDist notebook with minor contributions from myself and Romain Laine.

All StarDist models were trained on the same dataset, published as part of the ZeroCostDL4Mic project and created by Johanna Jukkala and Guillaume Jacquemet[199]. One parameter was chosen for fine-tuning to quickly identify an avenue for optimisation. The parameter deemed the most likely to affect learning during training was ‘patch size’. This parameter determines the dimensions of patches into which each image is divided, and thus, the number of patches for each image and the size of the objects in relation to each image. For users, this parameter is likely to be experimented with significantly as it gives additional flexibility beyond adjusting the learning rate or the number of epochs for training. Another parameter that users can change is the number of focal points per cell. However, it is likely that a larger number should usually lead to better segmentations. It will likely affect mostly the time used during training as the number of computations increases per cell. The relation of performance to patch size cannot be as easily predicted. As the depth of the network is not changed, it is possible that patches that are too small become difficult for the model to assess, as the representation maps become too small in the deeper layers of the U-Net architecture to identify features that could be upsampled (see U-Net architecture in Introduction). However, a smaller patch size may have the advantage that finer features could be identified for each nucleus as each object will appear larger in each patch. To investigate how this parameter may affect segmentation quality and the model's sensitivity, three models were trained with patch sizes set to 256x256, 512x512 and 1024x1024, respectively. Since the original image dimension was 1024x1024, the latter patch size creates only one patch per image covering the entire field of view. Each model was trained for 100 epochs (lr: 0.0002, foci: 32, batch size: 16).

The initial results suggested that the models were overtrained, i.e. every model overfit, with the largest patch size leading to overfitting setting on slightly later than in the models trained on smaller patch sizes. When visually inspecting the segmentation results for each model, only minor differences can be detected between the models (Fig. 33B). Overall, all models segment the nuclei well, with most ground-truth nuclei correctly detected in all models. Some differences can be seen in challenging areas of the image where nuclei are small, close together and touching (see insets in Fig. 33B). The model trained on the smallest patch size predicts images that show slightly more fringes around the nuclei, when overlayed with the ground-truth, suggesting a slightly less accurate segmentation result than in the 512x512 and the 1024x1024 patch sizes. Furthermore, this model does not detect a small dividing cell, suggesting a reduced sensitivity. The differences are slightly easier to detect when inspecting the models quantitatively using quality metrics on the full test dataset. The StarDist models are evaluated with object detection metrics (such as precision and recall) and the IoU for segmentation performance. Additionally, we use the *panoptic quality* metric which represents a combination of both[205]. Here, the IoU is summed over correctly identified (TP) objects and divided by a combination of TP, FP and FN (for details, see Methods). The most marked difference is the lower rate of FPs as the patch size increases, which appears to confirm the earlier visual observation. Yet, most other metrics vary little as they are all relatively high, with precision for all models above 0.9 and IoU above 0.8, suggesting generally good predictions. To ensure that problems did not arise from the overfitting of the models, the experiments were repeated with a lower number of epochs to stop training before overfitting occurred. However, models trained for fewer epochs do not significantly improve the performance of models trained for longer periods. Only the 512x512 patch size appears to show slight improvements in some metrics when trained for fewer epochs, but since the other models do not parallel this, it may be due to the randomness of the data allocated in the validation split or the initialisation of the model. By comparing all the models, the one with the highest scores in most metrics was the model trained for 100 epochs on the largest patch size. It should be noted here that the false positive rate in this experiment needs to be carefully evaluated because some of the predictions are clearly correct but are simply not labelled in the test dataset. Therefore, a small number of false positives may be expected. However, the number of false negatives remains relatively high compared to the false positives, suggesting that the sensitivity of the models could be further enhanced.

I chose to do this by augmenting the dataset and training again on the largest patch size to test whether the previous results could be optimised. When augmenting with purely geometric augmentations (rotation, flipping) the model performs better in all quantitative metrics although it overfits much earlier during training than any of the previously trained models. Visually, the main difference appears to be a slightly improved overlap between prediction and target. However, other differences cannot be clearly detected by eye. As the StarDist notebook provides further augmentations beyond those above, a further improvement of the model was attempted by adding augmentations that change the intensity of pixels on the image. This might make the model more robust against background noise or nuclei which appear dimmer or brighter on the images. However, the additional augmentation does not appear to improve the model beyond the performance of the ‘geometric’ augmentations.



C

patch size, epochs	IoU	FP	TP	FN	precision	recall	accuracy	f1 score	panoptic quality
256, 50	0.828	34	409	26	0.923	0.941	0.872	0.932	0.798
512, 50	0.835	19	**411**	**24**	0.956	0.945	0.906	0.951	0.804
1024, 50	0.76	51	406	29	0.887	0.934	0.836	0.911	0.72
256, 100	0.844	15	403	32	0.964	0.927	0.896	0.945	0.815
512, 100	0.842	11	401	34	0.973	0.922	0.9	0.947	0.811
1024, 100	0.843	9	409	26	0.978	0.941	0.921	0.959	0.814
8x augmentation (geometric)									
1024, 100	**0.864**	**5**	**412**	**23**	**0.988**	**0.948**	**0.937**	**0.967**	**0.838**
10x augmentation (geometric+intensity)									
1024, 100	0.852	6	407	28	0.986	0.936	0.924	0.96	0.826

Figure 33 – StarDist segmentation of breast cancer nuclei – A) *Training losses (mean average error) of StarDist models on 3 different patch sizes, with additional plots shown for training on an augmented dataset (8x) for the largest patch size.* ***B)*** *Example Test inputs and targets (left column) and predictions overlayed with the target from models trained on 3 different patch sizes and one model trained with augmentation. (Scalebars: full FOV: 100µm, inset: 50µm)* ***C)*** *Table of the quality metrics evaluated in the ZeroCostDL4Mic notebook. Highlights represent the best and second-best results for each metric. (IoU – Intersection over union, FP – False positives, TP – True positives, FN – false negatives)*

III. 4. QC for a denoising task in ZeroCostDL4Mic

While the evaluation of object detection and denoising tasks can be performed with metrics which are readily understandable and mirror the visual inspection of the images, evaluating denoising performance can be challenging as differences may be subtle and thus difficult to verify by eye. Here, I was interested if it was possible to evaluate and optimise CARE 2D models using the QC section in the ZeroCostDL4Mic notebook and how easy it was the estimate performance using three widely used metrics for measuring the performance of image restoration methods against ground-truth images: structural similarity (SSIM), mean squared error (MSE) and peak signal to noise ratio (PSNR).

The first trial for CARE 2D was performed on a dataset of 24 1024x1024 images of LifeAct-labelled HeLa cells. For QC, two images were set aside and each split into 4 patches of dimensions 512x512 pixels, which allowed testing on a slightly larger sample without reducing the already small dataset. As in the StarDist 2D notebook, one of the key parameters which can be adjusted in CARE is the patch size parameter. In the context of this project, this parameter has additional significance, as more RAM is required in the notebook when the patch size increases (at constant batch size). Hence, testing how patch size affects trained model performance could indicate the upper limit for model performance in this project.

For training, four different patch sizes with dimensions of 80x80, 128x128, 256x256, and 512x512 pixels were tested. A patch size of 1024x1024 was also attempted for training but could not be executed due to the RAM limitations of the Colab environment, making 512x512 the maximum patch size in this experiment. Since the patch size determines how many unique patches can be sampled from an image, the number of patches per image was reduced as the size of the patch increased. In CARE 2D, the number of patches per image is not automatically calculated as in StarDist 2D and is chosen by the user. CARE then creates random patches of the specified size and number for each image, which means that choosing a very large number of patches is likely to lead to oversampling of each training image. Although this may be problematic if overlap is too great as this could lead to overfitting, the authors of the CARE method do not report oversampling to lead to performance issues. Hence, for the above patch sizes, this was also done in this trial (see Fig. 34A), except for the

80x80 experiment where the 100 patches sampled from each image should not lead to significant oversampling, i.e. large overlap between two patches.

As above, the loss curves were evaluated first to identify overfitting models. The losses indeed indicate overfitting during training for all patch sizes, with the onset of overfitting delayed for larger patch sizes. While training losses are nearly identical for all patch sizes, the trajectories of the validation losses differ between the two smaller and larger patch sizes, with the former increasing clearly after overfitting while the latter continue to decrease (Fig. 34A), albeit more slowly than the training losses. Before attempting to overcome this problem, I was interested how much overfitting affected model performance on unseen data since experience from the StarDist 2D experiment showed that overfitting does not directly imply poor predictions, especially if model weights are saved based on the best validation loss. Comparing the predictions by the models on the test data no clear differences between models are apparent, with all models predicting images that fail to resolve some of the fine filaments seen in the ground-truth (Fig. 34B). Further investigation, using the error maps calculated in the notebook are also extremely similar, but may reveal subtle differences. The SSIM maps show relatively coarse regions of the image that differ between ground-truth and predictions. The main difference is that the models trained on the patch sizes between 128x128 and 256x256 appear to be overall brighter seen, for example, in the inset in Fig. 34B in the third row. The error maps calculated on the relatively simple root-squared error (RSE) between pixels are slightly more insightful than the SSIM for this dataset, showing quite clearly the filaments that are ‘missed’ by the models. The RSE maps also show that this error is more pronounced as the patch size used in model training increases. Since the error maps show only subtle differences, the same metrics were evaluated on the full dataset, to find which model performs best overall. Interestingly, the metrics suggest slightly better performance of the models on smaller patch sizes (higher PSNR and SSIM, lower NRMSE). While a patch size of 128x128 appears to yield the best performance on the test dataset in two out of three metrics, the test dataset is relatively small, and this result itself should thus be treated with caution. However, since the model trained on even smaller patches of 80x80 pixels achieves very similar performance and similarly performs slightly but significantly better than those trained on larger patches, this indeed suggests that smaller patch sizes help the CARE 2D model to perform better on unseen data.

How can this be reconciled with the overfitting behaviour of the models? First, it can be noted that only the best model weights are used for the testing step. This means that for the smaller patch sizes the weights used will be those which were learned before overfitting occurred, as validation loss increases after overfitting. Therefore, the model weights learned after overfitting are not saved. In contrast, validation loss decreases in models trained on larger patch sizes even after overfitting begins. This means that the best weights for these models are learned after overfitting to the training data. Hence, despite the lower validation losses and the later onset of overfitting, the best weights of these models may not lead to better generalisation than the models trained on smaller patches.

An additional observation which can be made when examining the results of the CARE 2D models on the test dataset is that the improvement of the images via denoising is small, as can be seen when comparing the differences between prediction and ground-truth and input and ground-truth.

The above results raise two questions: Firstly, can performance be improved for the larger patch sizes if training is stopped before overfitting? Since the above hypothesis is that the best model weights are those saved at the lowest validation loss, i.e. before overfitting, in the smaller patch sizes, using a model saved at even lower validation loss, but also before overfitting for 512x512, may further improve performance.

The second question is whether inherent features of the dataset limit the denoising performance. This question could be answered by training CARE 2D models on other datasets and determining if denoising performance as measured by the QC metrics is better on these datasets.

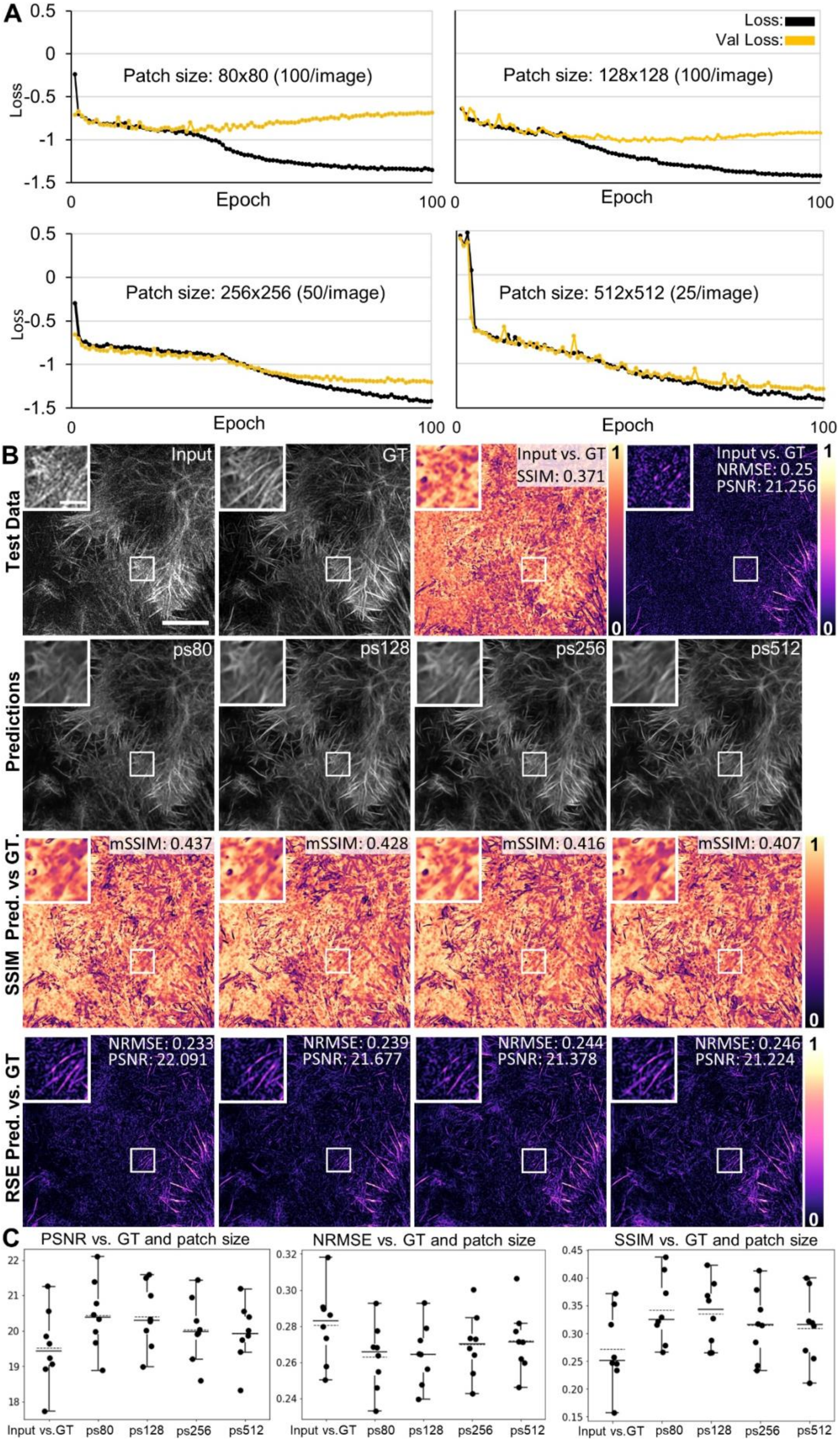

A
Loss:
Val Loss:
Patch size: 80x80 (100/image)
Patch size: 128x128 (100/image)
Patch size: 256x256 (50/image)
Patch size: 512x512 (25/image)
Loss
Epoch
0.5
0
-0.5
-1
-1.5
0
100
B
Test Data
Input
GT
Input vs. GT
SSIM: 0.371
Input vs. GT
NRMSE: 0.25
PSNR: 21.256
Predictions
ps80
ps128
ps256
ps512
SSIM Pred. vs GT.
mSSIM: 0.437
mSSIM: 0.428
mSSIM: 0.416
mSSIM: 0.407
RSE Pred. vs. GT
NRMSE: 0.233
PSNR: 22.091
NRMSE: 0.239
PSNR: 21.677
NRMSE: 0.244
PSNR: 21.378
NRMSE: 0.246
PSNR: 21.224
1
0
C
PSNR vs. GT and patch size
NRMSE vs. GT and patch size
SSIM vs. GT and patch size
22
21
20
19
18
0.32
0.30
0.28
0.26
0.24
0.45
0.40
0.35
0.30
0.25
0.20
0.15
Input vs.GT
ps80
ps128
ps256
ps512

Figure 34 – CARE denoising on actin using different patch sizes for training – A) *Loss curves for each patch size, with the number of patches per image shown in brackets.* ***B)*** *Example test data with predictions for each patch size and error maps for the SSIM and NRMSE metrics. The metrics shown in the upper right corner represent the average values for the shown image. (Scalebars: Full FOV: 5µm, inset: 1µm)* **C)** *Boxplot showing the average metrics evaluated for the whole test dataset (n=8) plotted against the patch sizes (in pixels per dimension) used for training (GT – ground truth, ps - patch size). (The whiskers encompass the range between the 1st and 3rd quartile of the distributions, with the median shown as solid and the mean as a dotted black line.)*

To answer the first question, the experiment above was repeated for the largest patch size, with checkpoints saved for the 20th, 50th and 100th epoch. For each checkpoint, the quality control steps were repeated.

To test if model performance deteriorates even when validation loss continues to improve, the models were trained on 512x512 patches for 20 and 50 epochs, to find a model which does not overfit. Fig. 35 shows that the models do not overfit at 20 or 50 epochs respectively, suggesting that the model trained for 50 epochs should contain the best loss achievable with the given set of parameters. As opposed to the previous trial, slight differences in quality can be seen between images from models trained for a different number of periods, with the model trained for 50 epochs being less grainy than the model trained for 20 epochs. Compared to the model trained for 100 epochs, which overfits to the training dataset, this model also appears to remove slightly less of the background structure and does not provide the same level of artificial smoothing. Subtle differences can again be seen on the error maps, with the model trained for 50 epochs showing slightly fewer dark (i.e. large error) areas than the other models in the SSIM map and more dark (low error) areas in the RSE map. Evaluating these metrics across the test dataset appears to confirm that training overfitting is a limiting factor and that 50 epochs seems to be near an optimal number of training epochs. This result shows that models perform better when the training is stopped earlier even if models trained for more epochs achieve a better validation loss (Fig. 35).

The results presented for this dataset show that in all cases the images predicted by the trained CARE method noise were closer to the targets than the inputs according to the quality metrics. The availability of the metrics also allowed performance to be improved even when the image quality was visually difficult to distinguish. However, models generally score low on all metrics with peak SSIM scores below 0.4, suggesting poor performance even if some models score high relative to others. This is also confirmed by visual inspection of the images themselves. The CARE 2D models predict images that are less granular than targets and appear to smooth edges, which can make structures visible in the ground-truth images completely disappear. Is this a result of the small size of this dataset or is the dataset inherently difficult to denoise? In the next section I explore this question by comparing the performance seen in the models trained on the small actin dataset with models trained on different datasets.

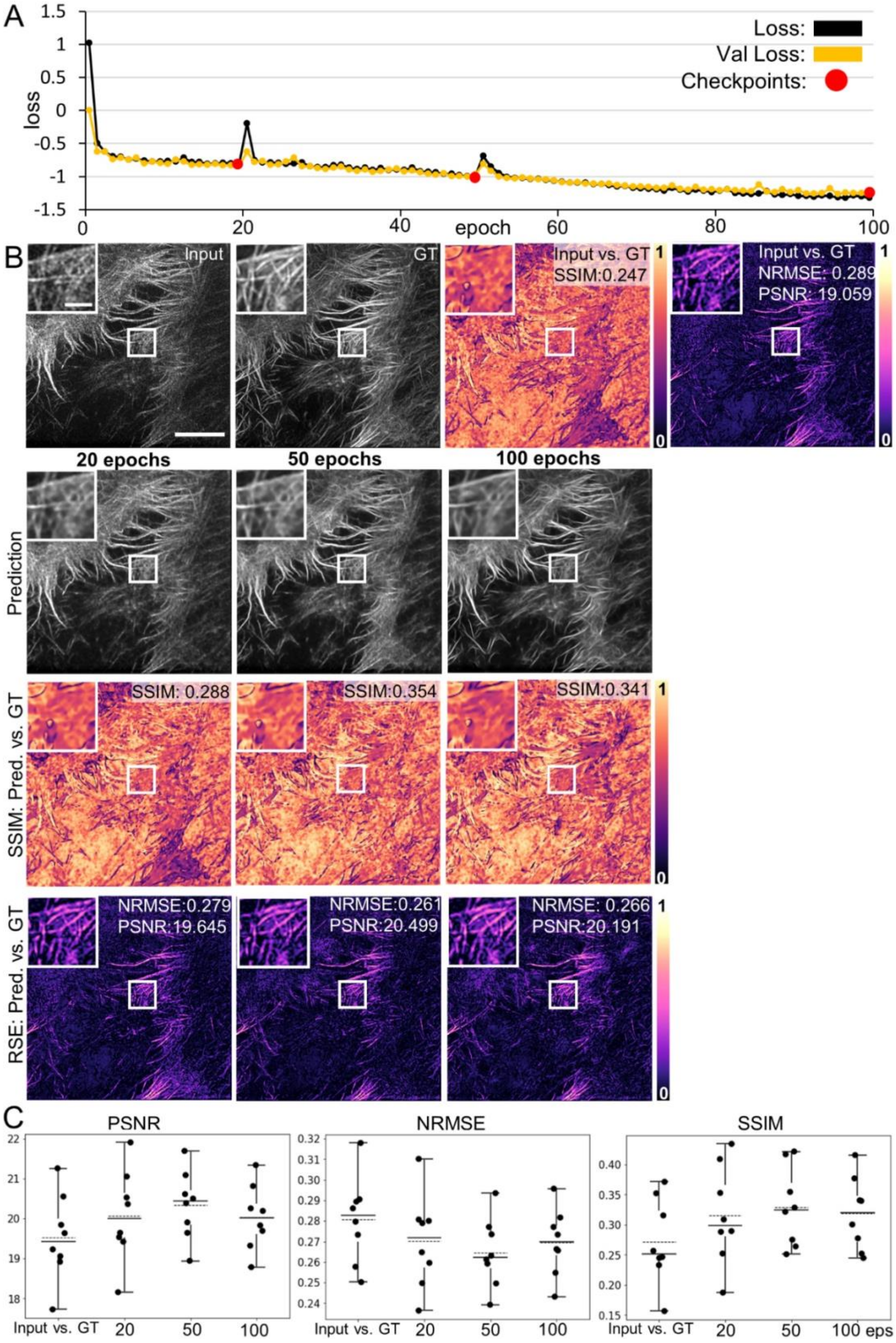


Figure 35 – Improving CARE performance by stopping training before overfitting* – A)** *Loss curves for model trained up to three checkpoints (patch size 512x512 pixels) on the LifeAct labelled dataset. The metrics shown on the top right refer to the average over the full FOV, not the inset. (Scalebars: full image: 5µm, inset: 1µm.)* ***B) *Example test images, predictions, and error maps for each checkpoint of the model trained in A.* ***C)*** *Quality metrics evaluated for CARE models on the above actin dataset for each checkpoint.*

While performance could be improved by adapting epoch numbers and patch size for training, the performance on the actin dataset remains low, with issues in reconstructing the filamentous actin structure even after fine-tuning.

To test whether there are inherent difficulties in denoising actin, CARE 2D models were trained on two other datasets, using parameter settings that yielded good performances on the actin dataset after 100 epochs (batch size 16, patch size 128, 100 patches, 100 epochs, 10% validation split). The datasets chosen were from HeLa cells labelled for mitochondria (labelled via TOM20 at outer mitochondrial membrane) and tubulin (using a fluorescently tagged β-tubulin subunit). Both datasets were significantly larger than the actin dataset used in the previous section (60 images for tubulin and 63 for mitochondria). The image dimensions were also adjusted so that the images were 1024x1024 pixels. To control for these differences in dataset size, training was performed both, on reduced versions of these datasets, with 22 training images, and on the full datasets. Alternatively, the actin dataset could have been augmented by 3, yielding approximately the same number of images as the other datasets (22x3=66). However, in that case, it would be the only dataset containing augmentations, which may not represent the full biological variability contained in a natural dataset. Furthermore, for testing performance by dataset, the scale of the dataset may not be of significance, provided that the models achieve at least minimal convergence.

The differences between the models’ performance on their respective datasets are immediately notable in the loss curves. The models trained on mitochondrial and tubulin labels converged smoothly without overfitting after 100 epochs (Fig. 36A). The performance also differs significantly when visually inspecting the models' output (Fig. 36B). The performance of the models denoising the mitochondrial and tubulin-labelled datasets offers a clear improvement without obvious errors. The error maps improve the input data compared to the ground-truth even clearer. In both mitochondrial and tubulin datasets, the difference in colour between the input-target and prediction-target pairs is clearly visible and reflected in both background and the labelled structures. This suggests that the improvement of the inputs upon denoising by CARE 2D is better for the tubulin and mitochondrial images than for the actin dataset.

These differences can also be quantified in the ZeroCostDL4Mic notebook. When examining the models' prediction on the full test datasets, the metrics suggest that the mitochondrial dataset achieves the best performance as indicated by the highest SSIM and PSNR and lowest NRMSE values (Fig. 36C). The model trained on microtubule labels reaches a slightly lower peak in these metrics, but still significantly outscores models trained to denoise the signal from the actin label. The high performance on the mitochondrial dataset may be partly explained by the higher similarity between the images in the training pairs, as lower discrepancies between these pairs likely make it easier to learn this task. However, this is likely not the only reason for the improved performance, as the microtubule training pairs have a similar discrepancy as the actin training pairs, according to the three quality metrics. The performance was pronounced for datasets of identical dataset size, and as expected, performance further improved, although only slightly, when trained on the full tubulin and mitochondrial datasets. This minor improvement upon a nearly 3-fold increase of the dataset suggests that increasing the dataset size for the actin dataset, which was not possible as no further images could be acquired, would not have significantly boosted the performance of the model trained on this dataset. The poor performance on the actin data is thus unlikely to be purely a result of its limited size.

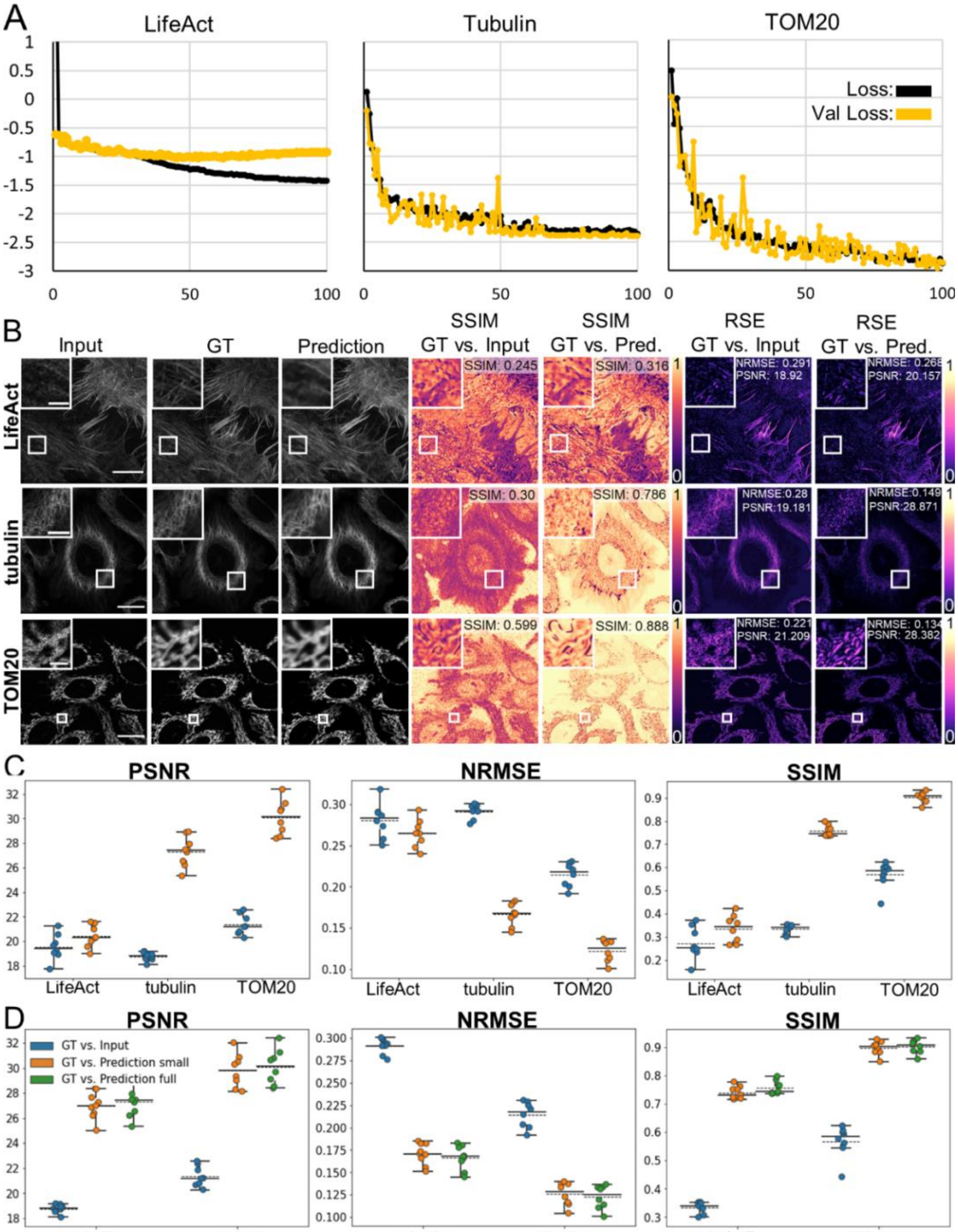

A
LifeAct
Tubulin
TOM20
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
0
50
100
Loss:
Val Loss:
B
Input
GT
Prediction
SSIM GT vs. Input
SSIM GT vs. Pred.
RSE GT vs. Input
RSE GT vs. Pred.
LifeAct
tubulin
TOM20
SSIM: 0.245
SSIM: 0.316
NRMSE: 0.291 PSNR: 18.92
NRMSE: 0.268 PSNR: 20.157
SSIM: 0.30
SSIM: 0.786
NRMSE:0.28 PSNR:19.181
NRMSE:0.149 PSNR:28.871
SSIM: 0.599
SSIM: 0.888
NRMSE: 0.221 PSNR: 21.209
NRMSE: 0.134 PSNR: 28.382
1
0
C
PSNR
NRMSE
SSIM
LifeAct
tubulin
TOM20
D
GT vs. Input
GT vs. Prediction small
GT vs. Prediction full
PSNR
NRMSE
SSIM
tubulin
TOM20

Figure 36 – CARE models perform differently on different datasets – A) *Loss curves for three different datasets evaluated with a patch size of 128x128.* ***B)*** *Example images, predictions, and error maps for test datasets for CARE 2D models trained with three different labels, with a training patch size of 128x128. The metrics in the top right corner are the average values for the whole FOV shown. (Scalebars: LifeAct full FOV: 5µm, inset: 1µm, β-tubulin and TOM20: full FOVs: 20µm, insets: 5µm)* ***C)*** *Average quality metrics evaluated for entire test datasets for actin (n=8), tubulin (n=8) and mitochondria (n=9).* ***D)*** *Comparison of quality metrics results in C (here in orange) with models trained with larger datasets (green), small datasets: 22 training pairs, full datasets: TOM20 – 63 training pairs, tubulin – 60 training pairs.*

Hence, there may be an inherent difficulty in denoising the actin dataset instead of a problem related to the dataset's size. However, these results do not indicate whether the inherent difficulties in denoising arise from the structure depicted on the images themselves, i.e. the actin cytoskeleton, or if they result from the type of noise in the data which may be difficult to remove.

The noise level can be artificially controlled to assess whether the noise itself rather than the biological structure may contribute to these difficulties. To control the noise, new datasets were created where 10% Gaussian noise was added to the target images from all three datasets that were previously normalised to values between 0 and 1. CARE 2D models were then trained with the 'noised' images as inputs and the normalised targets. Suppose the differences in performance initially arose primarily due to differences in noise between different datasets caused by differences in the acquisition conditions, rather than differences inherent in the acquired biological structures. In that case, these should disappear when the noise is similar in all images. However, the models trained on these artificial denoising datasets continue to perform poorly on the actin-labelled dataset when compared to both mitochondria and microtubule labels (Fig. 37A). This supports the hypothesis that the difficulties are inherent in the image content rather than in the statistical distribution of the image noise.

However, another interesting conclusion can be drawn from the results of this investigation. As the input images are all denoised to the same degree, it could be expected that the QC metrics would show almost identical results for all three datasets. This is the case for NRMSE and PSNR, both of which fundamentally calculate a pixel-wise error on the images (Fig. 37B). However, the SSIM metric appears to record a clear difference between the actin dataset and the other datasets. Inspecting the error maps suggests that this effect could be caused by the relative ratio of background and foreground on the images, with background areas consistently brighter than 'structure' on the image. Although it is difficult to quantify this as the allocation of 'foreground' or 'background' could introduce bias into the analysis, the colour of the structures in all datasets is similar, suggesting that denoising performance is not different between structures. However, images differ in terms of their structure content, which may explain the differences in SSIM between inputs and targets, which are smaller for the actin dataset than for the others.

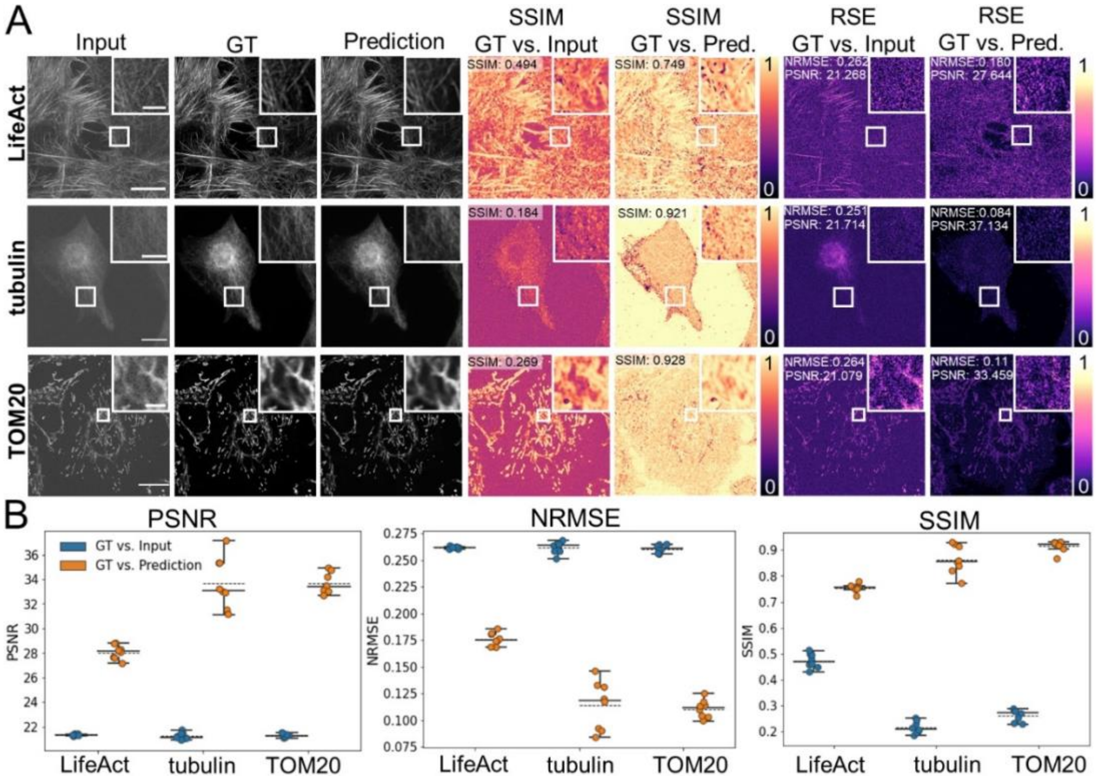


Fig. 37 – CARE control with artificial noise – A) *Example for test images, predictions, and error maps for models trained on three different datasets. (Scalebars: LifeAct full FOV: 5µm, inset: 1µm, β-tubulin and TOM20 full FOVs: 20µm, inset: 5µm.)* ***B)*** *Boxplot of the average quality metrics for the full test datasets for each label.*

Since this effect does not depend on the predictions of the CARE model, it could be verified on a larger sample of images by introducing the same noise as above, but to the training datasets.

Here, the QC section in the notebook was slightly adapted to allow comparisons between images without a model for predictions by simply commenting out the code lines used to calculate metrics on a model's prediction (see Fig. 20 in chapter II.). When QC is performed on the full training datasets between inputs and targets, the results of the smaller sample appear to be confirmed (Fig. 38). While the PSNR and NRMSE scores show that the noise leads to a near-identical measured difference between the images, the SSIM score for images containing the LifeAct label shows the same bias as seen above, likely due to the higher fraction of foreground signal which SSIM inherently measures as more similar between inputs and targets than background. This experiment also showed that the QC section can be used independently from the DL methods implemented in the notebooks, to investigate datasets, without intermediary DL models. Indeed, the QC methods for image comparisons are now also available as an independent notebook, created by the Jacquemet group.

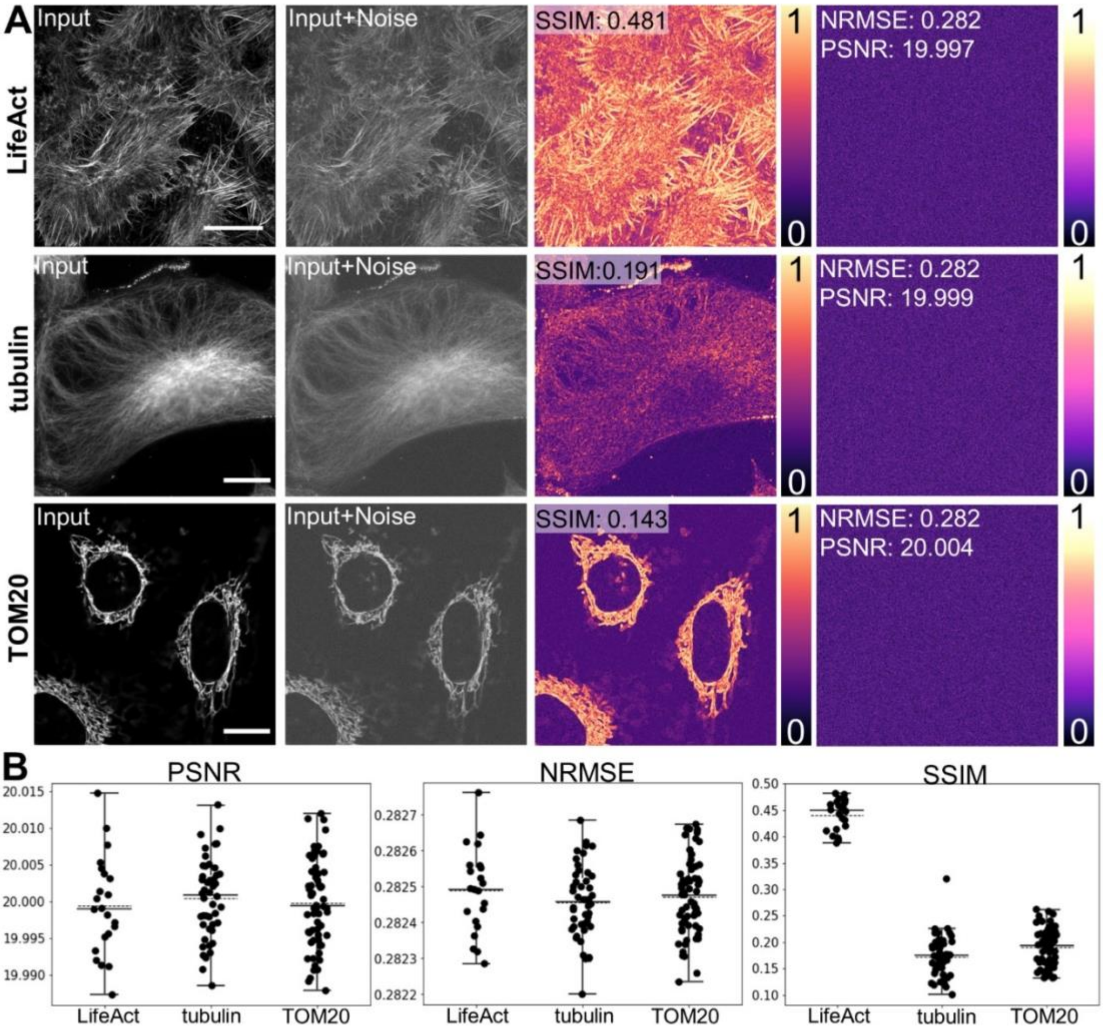


Figure 38 – CARE and quality metrics on images with noise controls – A) *Example images, predictions and error maps from test datasets for three different labels, with input images obtained by adding Gaussian noise to the ground-truth (GT) images, metrics refer to the average metrics evaluated over the full FOV. (Scalebars: LifeAct: 10µm, β-tubulin and TOM20: 200µm)* ***B)*** *Average quality metrics evaluated over the full test datasets (prediction vs. GT), for three different labels for models using input images with added Gaussian noise.* ***C)*** *Target images from the training datasets of the three different labels, the resulting noisy image after Gaussian noise is added, and error maps for the difference between these images.* ***D)*** *Average quality metrics evaluated over the full training datasets after all targets had Gaussian noise added. The metrics compared these noisy inputs with the original ground-truth images.*

III. 5. QC for an artificial labelling task in ZeroCostDL4Mic

In label-free prediction, model training is implemented differently to the other methods in the ZeroCostDL4Mic project. Instead of dividing the dataset into batches and letting the neural network repeat gradient descent on the same set of batches over multiple epochs, the dataset in fnet is divided into many batches each of which the model 'sees' only once during training. For each step, a batch is created from patches sampled from the images in the dataset. Using the *batch_size* parameter the user can choose how many patches are contained within each batch. Since each batch contains a unique set of patches from the dataset and the number of steps determines how many batches are used for training, the patch size and batch size should be set reasonably to avoid excessive oversampling. However, in most DL-methods, the neural network will encounter each training pair once per epoch, and this may be repeated over 100 times (or more), without necessarily leading to overfitting models, as seen above. Hence, when training fnet which is based on the U-net architecture as in StarDist and CARE, the dataset can likely contain some redundancy without resulting in an overfitting model. Here, the following assumptions were made to determine an initial set of parameters to test how patch size can affect the models' performance.

First, it was assumed that since other models can be trained to roughly 100 epochs without overfitting on smaller datasets than the one used here (83 images for training, 9 for validation, dimensions of 512x512x32) that oversampling images by a factor of 100 should be equivalent to training a model for 100 epochs, as long as each sample is 'seen' only once. Given that batch size is limited by Colab's RAM and that larger batch sizes require longer training times, I decided to use a batch size of 2 for the largest patch size I wanted to test (256x256x32). Given that there are 4 unique patches for an image of size 512x512x32 for this patch size and allowing oversampling of 100 (see above), I calculated the number of steps to be used during training as follows:

$$steps = \frac{\frac{patches}{image} * 100 * images}{batch\_size}$$

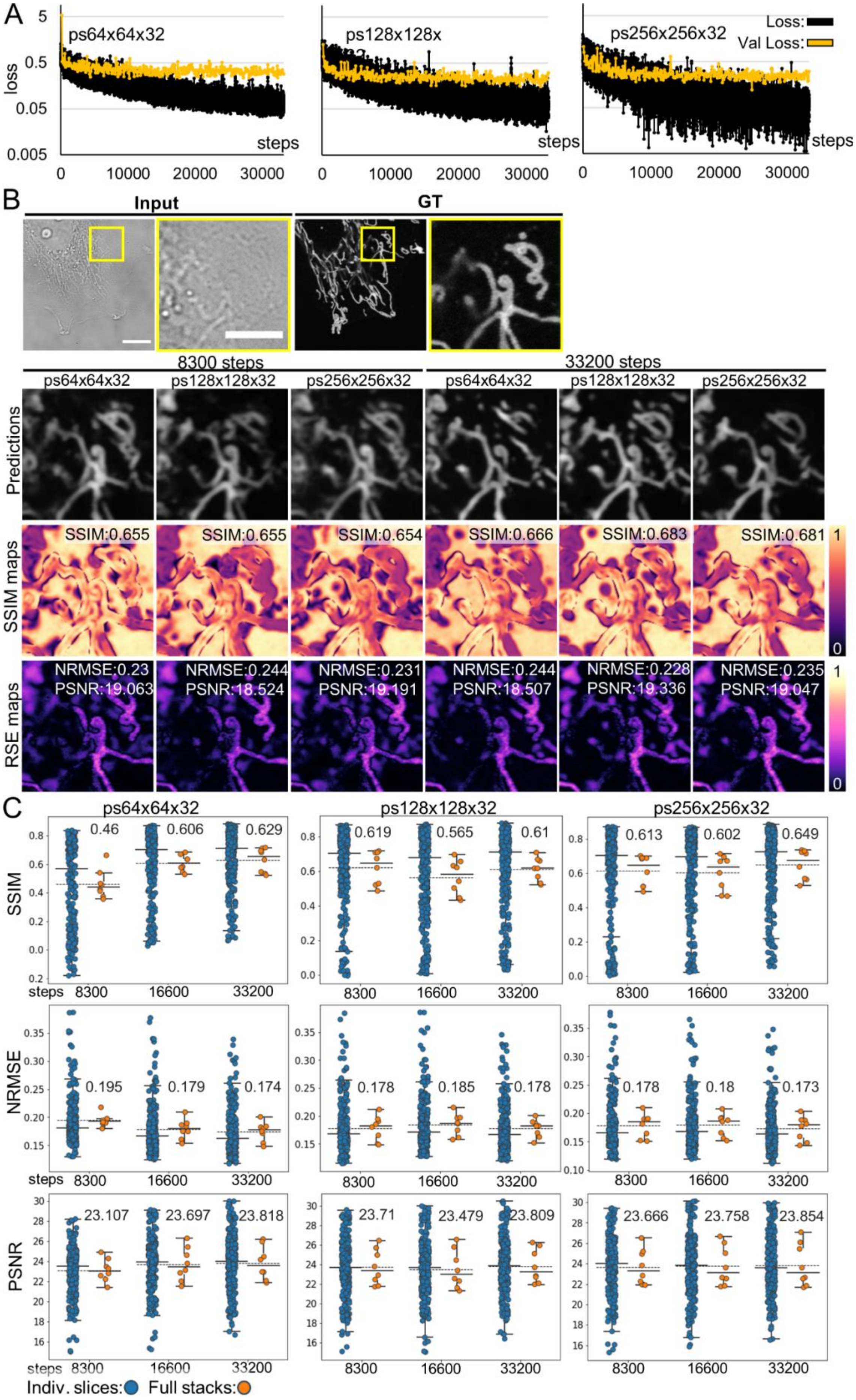
For the above parameters, this gives a step number of 16.600. Keeping the number of steps equal for the other *patch_sizes* and using the same considerations regarding the number of patches per image for the patch sizes of 128x128x32 and 64x64x32 can be achieved by changing the batch size. Using the above formula and using the same number of steps, gives batch sizes of 8 and 32, for the above patch sizes, respectively. Using these parameters, several fnet 3D models were trained. In addition, to increase the likelihood of observing quality differences between models, further models were trained with lower (8300) and higher numbers (33200) of steps, while keeping all other parameters stable. This should only increase the number of samples picked from the datasets and should therefore be equivalent to a larger number of epochs in other models. The first notable observation from the training of these models is that for step numbers higher than 8300, all models overfit, with the validation loss stabilising after roughly 5000 epochs for all patch sizes, and the training loss continuing to decrease. As the patch size grows larger the training losses initially decrease slower, and with more variation, eventually reaching lower values at the end of training than models trained on the lower patch sizes. Validation losses while stable in all models are slightly lower for the batch sizes of 128x128x32 and 256x256x32 than for the 64x64x32 patch size (Fig. 39A). These results suggest that the model does not perform well on this training dataset and does not appear to improve performance significantly between patch sizes according to validation loss.

This is partly confirmed when examining the models' predictions on the test dataset. Yet, on first sight, all models produce visually compelling predictions which show many of the structures present in the ground-truth which would likely be difficult to predict by eye (Fig. 39B). In this respect, the model succeeds in predicting fluorescent signals. However, upon closer inspection and using the error maps produced in the QC section, the predictions show clear artefacts in the form of signal not present in the ground-truth. For the larger two patch sizes and the largest step number, there is some indication that the models predict fewer artefacts, with slightly improved SSIM scores (0.683 for 128x128x32 at 33200 steps vs. 0.655 for the same patch size at 8300 steps). However, these improvements are minor and much of the prediction performance appears to be a result of correctly predicting the background rather than a signal of the structure of the mitochondrial TOM20 label (Fig. 39B).

In terms of content, the SSIM maps again show that background areas appear to be generally well predicted by models, appearing much brighter than the areas containing structure. The SSIM maps clearly highlight areas with discrepancies as darker regions, while correctly predicted structures are shown a lighter colour. However, the SSIM maps appears to enlarge these regions around their edges which helps to highlight small errors but simultaneously blurs out some of the details in more crowded areas. In structures that appear well-predicted the SSIM maps also reveal that the fnet 3D models appear to slightly mis-predict the edges of the mitochondrial structure which is depicted as dark edge around the overall correct location of the structures. The RSE maps show the misrepresented areas with relatively more precision, making incorrect predictions more clearly demarcated than in the SSIM maps and making smaller artefacts more challenging to spot.

On the full test dataset, the model's performance can be evaluated on the full test stacks, i.e. the average of the errors on all predicted slices of the stacks or by the individual slice. When measuring the performance along with both metrics, it becomes clear that none of the models clearly stands out in terms of performance on the test dataset. Only small differences are evident as step number increases, with all metrics measuring small increases for an increasing number of training steps (Fig. 39C). For the patch size of 64x64x32, this improvement is most consistent. It is most clearly resolved by the SSIM metric, which shows distinct differences between models trained for a different number of steps. In contrast, the other metrics do not resolve the differences as distinctly. The differences between patch sizes are subtle, and models trained on larger patches scored higher than those trained on smaller patches. These results suggest that despite overfitting, the smaller training losses at the end of training for larger patch sizes correlate to a slightly better prediction performance on unseen datasets.

A difficulty in training fnet in notebooks is that it takes significantly longer to train a single model than in all the other notebooks. At 33200 steps, depending on the allocated GPU of the Colab notebook, training often exceeded 10 hours of training, and models had to be re-trained when the runtime disconnected in the training time. Hence, increasing the step number, leading to better models, becomes difficult in the notebooks.



A
ps64x64x32
ps128x128x
ps256x256x32
Loss:
Val Loss:
loss
5
0.5
0.05
0.005
steps
0
10000
20000
30000
B
Input
GT
8300 steps
33200 steps
ps64x64x32
ps128x128x32
ps256x256x32
Predictions
SSIM maps
SSIM:0.655
SSIM:0.655
SSIM:0.654
SSIM:0.666
SSIM:0.683
SSIM:0.681
RSE maps
NRMSE:0.23
PSNR:19.063
NRMSE:0.244
PSNR:18.524
NRMSE:0.231
PSNR:19.191
NRMSE:0.244
PSNR:18.507
NRMSE:0.228
PSNR:19.336
NRMSE:0.235
PSNR:19.047
1
0
C
SSIM
0.46
0.606
0.629
0.619
0.565
0.61
0.613
0.602
0.649
NRMSE
0.195
0.179
0.174
0.178
0.185
0.178
0.178
0.18
0.173
PSNR
23.107
23.697
23.818
23.71
23.479
23.809
23.666
23.758
23.854
steps
8300
16600
33200
Indiv. slices: ●
Full stacks: ●

Figure 39 – Artificial labelling using label-free prediction (fnet) and training using different patch sizes and for different numbers of training steps – A) *Loss curves for fnet models trained on different patch sizes (displayed above the curves).* ***B)*** *Example of test images, predictions, and error maps for fnet models trained on different patch sizes and numbers of steps on HeLa cells containing a mitochondrial label (TOM20). The full FOV is shown only for the test input and targets. The quality metrics in the top right corners refer to the full FOV, not the inset. Only the first slice of the full stack (z=32) is shown. (Scalebars: Full FOV: 10µm, inset: 5µm)* **C)** *– Average quality metrics evaluated for the full test dataset (n = 8) for fnet models trained for different patch sizes and number of training steps. The plots show both the evaluation per slice in the image stacks (blue) and the average score per stack (orange).*

In the publication, the authors of fnet trained their models for 250,000 steps for the reported results (although they also use a larger dataset)[114]. Since performance improvement was marginal for the tested step numbers, one further trial on the largest patch size was performed with 100,000, which likely represents somewhat of a limit of what is a realistic use-case of the Colab environment for small to medium bioimaging projects. Training for this model required several runtimes and took five days to be trained using a free Google account. The losses for this model diverge slightly later than seen in the other models, but again suggest overfitting. However, training losses tend lower towards the end of training than in any previous model (Fig. 40A). Visually, the model predictions are slightly brighter with sharper edges around predicted structures, for both correctly predicted ones and artefacts, leading to a slightly ‘cleaner’ predicted image (Fig. 40B). However, the performance of this model as suggested by the error maps and metrics calculated on the example test image only shows a small improvement in SSIM and no improvement in the other metrics (NRMSE and PSNR) compared to the set of the models trained before (Fig. 40B). When evaluated on the full dataset, the quality metrics similarly do not suggest a significant improvement although the model reaches the best average PSNR, second best average SSIM and the joint lowest average NRMSE score (Fig. 40C). This suggests that additional training can enhance the predicted signal over the background noise, which may corroborate the impression from the prediction, but does not offer a significant improvement in terms of detected structure or precision.

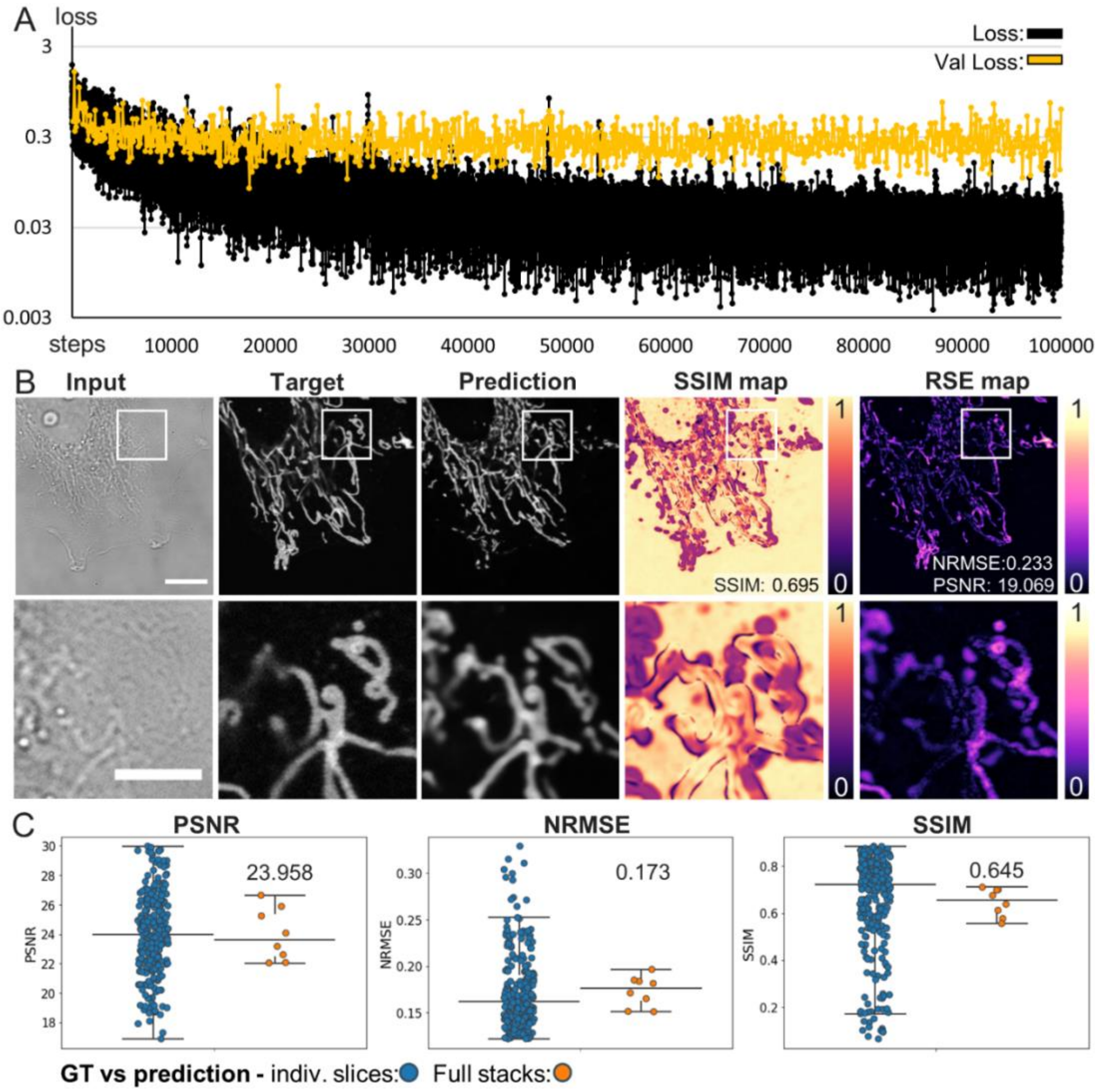


***Fig. 40 – Does performance in fnet improve when the training step number is increased? – A**) Loss curves for the fnet model trained on the dataset from Fig. 39, for 100000 steps. **B)** Example test images from the test dataset (n = 8), same as in fig. 39., with only the first slice of the stack shown. Quality metrics refer to the full FOV. (Scalebars: Full FOV: 10µm, inset: 5µm) **C)** Average quality metrics evaluated for the full test dataset for slices per stack (blue) and averages for the full stacks (orange).*

III. 6. Discussion

Most published DL methods give little advice to users on how to assess the quality of trained models, and few provide methods which can be easily used to perform QC. However, QC is crucial to achieving the best DL performance and is inevitable if these tools are used in research. Since the reliability of DL is central to its trust and use in the community and one of the goals of this project was to improve access to reliable DL methods, it became central to the project to integrate QC as a step in the workflow. This section explores the merits of the QC metrics available for the models trained in ZeroCostDL4Mic. This was necessary to demonstrate that the platform introduced in the previous chapter could tackle a serious problem hindering the uptake of DL methods in bioimaging, the 'reliability problem'. While this aim acted as a starting point for this chapter, several related questions could be answered in its pursuit, and additional questions arose which will continue to inspire further work. The first key question explored in this section was to which degree the models trained on the platform can fulfil their intended functions. Can the QC section of the notebooks show if the models learn the tasks they were designed and trained for?

As the first element of the QC the user will access in the notebooks is a graphical representation of the losses, validation losses and (for YOLOv2) mAP scores accumulated during training, this was also the first step in analysing the quality of the models trained here. The first immediate observation was that all models showed overfitting to the training datasets to some degree. However, when analysing the models' predictions and comparing them with the ground-truth via visual inspection of error maps and quality metrics, this does not mean that the models could not be trained for their respective tasks. Indeed, all the methods could be used for their respective tasks and evaluated using the QC section. Not all models performed equally well but the QC section tended to flag weaknesses. The most helpful aspect of the QC section in this regard were the error maps as these can highlight features that are not captured using quality metrics. For example, this helped identify cells that were not labelled in the ground-truth in the StarDist segmentation dataset which under the use of quality metrics only would have been highlighted as a 'false positive' even though models correctly predicted and segmented the nucleus. In the denoising task, the error maps for the SSIM and RSE scores highlighted areas of the images that were not correctly predicted, either via artefacts or missed objects, specifically for the actin dataset, which appeared to represent a difficult case for the CARE models trained here. Fnet also performed its task relatively well even though training was limited by its time to train individual models. Visually, it performed relatively well in extracting the mitochondrial structure from the brightfield images. However, it failed to resolve details in mitochondrial structures and predicted artefacts, the identification of which was helped by the error maps. For the YOLOv2 notebook, error maps are not provided in the notebook. This was a deliberate choice when designing the notebook, as the overlay of bounding boxes from different images can be confusing, especially when many objects are found in an image. Hence, predictions and ground-truth labels are shown side-by-side which was sufficient to identify the weaknesses of the trained models. Here, the quality metrics actually were more insightful as the quantification of mistakes made by the models could be evaluated with more clarity than by inspecting the images by eye.

The second question explored in this section was whether the metrics could help to fine-tune the models trained through the platform, thus optimising them for specific tasks for which parameter changes or dataset adjustments may be required.

As mentioned above, an apparent problem for models trained for this chapter was a tendency to overfit. This problem acts as a good first target for optimisation of model training. Indeed, for almost all methods, the parameters or options in the notebooks could be used to mitigate this problem, most notably in CARE and YOLOv2 where early stopping and augmentation helped prevent overfitting during training and improve model performances as measured by quantitative metrics, respectively. The observation that overfitting could be compensated for in YOLOv2 by augmentation is an indication that this problem may arise from the small dataset sizes used here. In the other models the reasons for overfitting were not as clear. While dataset size may be the main underlying cause, there may be other reasons as well, especially since methods such as CARE and fnet were shown in publication to perform well when trained on datasets with comparable sizes[113,114]. In the CARE 2D and StarDist 2D notebooks, overfitting behaviour changed when the patch size was changed, with a larger patch size usually leading to less overfitting during training. In contrast, in fnet 3D, patch size changes did not improve validation loss, the reasons for which are unclear. However, it should be noted that in the original publication, validation is not implemented in fnet by default, and it is also not shown in publication, which suggests that the model may have overfit in the original publication as well and yet performed well on unseen data. Indeed, when the models for the three methods were tested on an unseen QC dataset in the notebooks, validation loss was not always an indicator for performance measured by the quantitative metrics. In CARE 2D, performance deteriorated with a larger patch size despite reduced overfitting. However, it was noted that while overfitting was less pronounced, the validation loss continued to decline. Because model weights are saved each time the validation loss decreases, and he lowest validation losses occur after overfitting, this may explain why model performance may be lower for these models (trained on 256x256 and 512x512 patch sizes. Indeed, when training was stopped for this patch size before overfitting, performance according to quality metrics, improved. This showed that overfitting can take two forms: validation loss continues to 'improve' after overfitting, likely leading to models that perform worse on unseen data, and one in which validation losses remain stable or increase after overfitting. In the latter case, since only the lowest-validation-loss-weights are saved, this form of overfitting may not necessarily reduce performance of the trained model which is suggested by the CARE 2D model trained on a patch size of 128x128, which performs better than the model trained on 512x512 patch sizes despite overfitting quite dramatically. Training may be less prone to overfitting on a larger patch size may be related to the nature of actin in the dataset. As the patch sizes increase more details of the relatively fine actin filaments remain after down sampling the patches in the neural network of CARE, leaving more information to be learned, while this information may disappear for smaller patch sizes.

In StarDist 2D, overfitting unexpectedly increased with augmentation. It is not clear what causes this behaviour, but the input images may contain some unique features that are amplified by augmentation and that are not representative of the full data diversity. However, the reasons for this were not pursued further, as the performance of the model trained on the augmented dataset was very good in terms of segmentation accuracy and detection sensitivity, improving over models trained on smaller patch sizes. Again, the overfitting on the dataset appears to be of the type that leaves validation losses constant. The weights leading to the best validation loss are saved before overfitting, as in the above CARE 2D models.

The observations on training losses and their effects on model performance as evaluated using different quality metrics allows two preliminary conclusions about the ZeroCostDL4Mic platform. First, models can indeed be trained in the ZeroCostDL4Mic notebooks to converge on relatively small datasets. In the case of StarDist 2D, performance on the test data was so high after augmentation that it is unlikely to increase much beyond even after further training. The second result is that all models showed improvements in parameter changes that could be identified in losses during training and performance evaluation using QC metrics. This means that the features implemented in the notebooks can indeed be used to fine-tune models via dataset or parameter adjustments, which was one of the key aims of this section.

Yet, there were also shortcomings and differences in the value of the quality metrics for different tasks, which raise some questions about using some of these metrics for evaluating model performance. For tasks based on object detection and segmentation challenges, the widely used TP, FP, and FN measure image- and dataset-wide metrics which give an accurate evaluation of objects which can be easily confirmed by visualisation of examples. For YOLOv2, the evaluation of these metrics in the QC section clearly showed which models performed better on detecting certain classes, which could be quickly confirmed by displaying instances of the predictions side-by-side with the ground-truth. This also allowed identifying class imbalance as a potential cause of difficulties, as adjustments in the number of classes per training clearly improved performance both visually and in the quality metrics. Similarly, though perhaps less clearly than in the YOLOv2 models, the StarDist evaluation of TP, FP, FN and IoU provides intuitive insights into how well the model performs across entire images and datasets. For instance, missed objects or inaccurate segmentations are immediately visible by analysing the metrics and by eye in the target and prediction overlay maps. The quality metrics are readily interpretable for these tasks and provide reliable insights into model performance. Indeed, these metrics can be used for evaluating higher-level metrics such as F1 and mAP which make different models comparable with single metrics. Because these metrics provide relatively accurate estimates of model performance, they have also been widely used in object detection and segmentation challenges in the computer vision community and are optimised for these tasks.

This stands in contrast to the image quality metrics SSIM, NRMSE, and PSNR for models that reconstruct full images, such as CARE and fnet. These metrics have been used frequently in tasks for image reconstruction in bioimaging, and in the results from the QC section, they suggest that the denoising and artificial labelling models appear to improve their performance according to some intuitive parameter changes discussed above. However, in terms of interpretability these metrics are far less insightful than their counterparts for object detection and segmentation. For example, can it truly be concluded from the evaluation of the models in figures 34-35 that the predictions from the CARE 2D models on actin represent successful removal of image noise or object reconstruction from the input images? The metrics suggest a small improvement of the predictions when compared to the input images. However, these improvements are not clear from inspecting the predictions of the model which appear to show smoothing and a lack of detail for very fine details of the actin cytoskeleton. Therefore, the predictions themselves suggest that the denoising performance is not as good as intended, making the evaluation by the metrics difficult to interpret.

Indeed, all metrics used in the QC for image reconstruction methods measure features other than denoising, namely the pixel-wise error between two images (NRMSE and PSNR) or the similarity of structures in the images (SSIM). SSIM likely has more value here as denoising tasks are inherently more concerned with structural information than pixel-wise precision e.g. in prediction of intensities. This can be seen when artificial noise is added to ground-truth images. Structures are clearly still present in the image containing noise, the MSE metric does not consider the similarity of these structures as most of it is obscured by pixel-wise noise and gives a relatively high error for the whole image.

The observations suggest that these metrics are poor predictors of denoising performance by themselves. However, the shortcomings can be compensated or interpreted by the availability of each metric's error map in the ZeroCostDL4Mic notebooks. These clarify some of the issues but also the strengths of the metrics, which are not apparent from the values alone. In the case of the actin dataset, a clear problem from inspecting the predictions was that the model performs poorly on unseen images. While the image metrics that are calculated across entire images suggest a small image improvement, the SSIM map clearly shows how artefacts were introduced and the RSE map, which forms the basis for the NRMSE metric, highlights actin filaments that were not predicted. The error maps thus nicely complement each other in showing issues with the predictions. Yet, on average all metrics measure an improvement. How is this explained? This becomes clearer when comparing the performance of the CARE 2D model on actin with that of models trained on different datasets.

When comparing noisy input images to targets, SSIM, NRMSE and PSNR appear to identify structures, i.e. areas where signal dominates noise on images very well, and gives low (SSIM) or high (RSE) values for background regions, likely because the structure of the noise is very unlike the mostly structureless background area in the ground-truth image. In contrast, when the image is denoised and compared to the target, SSIM in background regions is extremely high, and RSE low, likely because the model easily predicts background signals as dark areas, while the structured areas see a smaller improvement. These attributes of SSIM and RSE mean that they are biased towards datasets with more background areas and make the metrics across whole images challenging to use as the sole indicators of denoising performance. While this problem is present for both metrics it appears to be more pronounced in the SSIM metric, likely because it is designed to identify 'structure' which is inherently more difficult to reconstruct via denoising or artificial labelling than background regions. This is one reason why the actin dataset, which has very little background per FOV, stands out when virtually identical noise is added to it as to the other investigated datasets depicting microtubule and mitochondrial labels, and SSIM appears less sensitive in highlighting the differences between low-noise and high-noise instances.

Despite the conceptual challenge in interpreting what the image quality metrics measure, they do appear to provide an indication of model performance which matches inspection of the predictions and error maps by eye. Therefore, the metrics suggest that the CARE 2D models do not denoise actin, tubulin and mitochondria, which is reflected in the overall lower scores between the prediction and the targets for this dataset than for the other datasets. While factors such as background and foreground need to be considered when evaluating the values of SSIM which is helped by error map consultation, when models need to be fine-tuned for a specific dataset, the average metrics can help to identify patterns in performance when visual inspection fails to find significant differences. For example, if different models are used on the same image, and the background is interpreted roughly in the same way by different models, then differences measured by SSIM or NRMSE do correspond to differences in the quality of structural predictions.

For the case of fnet 3D, it was virtually impossible to discern the differences between model performance for different parameter settings by eye alone, even when the error maps were considered. Yet, when the full test dataset was evaluated small but clear differences between the models emerged which helped in optimising the training in the notebook.

The results from this chapter show that the main quality metrics used for DL-methods in image reconstruction do not carry the same informational value as those used for object detection. The reasons for this may lie in the origin of these metrics and the tasks they were designed to analyse. While the metrics for object detection are intuitive and relate essentially to a binary problem (objects are either correctly classified/detected or not), they also do not differ between macroscopic or microscopic objects. For image reconstruction problems this equivalence does not hold. MSE, PSNR and SSIM are metrics developed essentially to determine how image data compression deteriorates image content and has been tested primarily on macroscopic images in computer vision. In macroscopic images, the differences between background and foreground are usually not as pronounced and not as relevant as for microscopic images, specifically fluorescent microscopy images. The problem here lies in the fact that all methods create an average metric across entire images when a distinction between fore- and background is crucial to determine the performance of the algorithm. The metrics also do not have a ‘count’ for missed objects or introduced artefacts which would be more akin to the FP or FN metrics in object detection or segmentation tasks. In this project, the metrics which were implemented in the QC section, and which were shown here were chosen because they play a role in democratisation. Since these metrics are widely used in the community, they represent a 'standard' that will help users put any results from the platform in perspective. However, this standard should perhaps be scrutinised and adjusted to reflect the needs of the bioimaging community, which are distinct from those of standard computer vision tasks. In this project, the best solution to the problem with some QC metrics was to display them via pixel wise representations on the error-maps. Currently, for image reconstruction methods, this appears to be the safest option for users to judge whether their models can be used safely on unseen datasets. The alternative, using the metrics themselves as is commonly done for DL-methods in bioimaging cannot be deemed as a suitable quality control for these methods from the observations in this chapter.

However, it is not inconceivable that metrics exist which are better predictors for model performance and can be more intuitively interpreted without additional human visual confirmation. In the cycleGAN and pix2pix notebooks, also part of the ZeroCostDL4Mic project, the LPIPS metric[206] is implemented on an experimental basis, as generator output may be of any type of data, which makes such outputs difficult to assess with existing methods such as SSIM and NRMSE. LPIPS uses the weights of a pretrained neural network trained to mimic humans in determining the differences between two images which has previously been shown to perform better in certain image-to-image tasks than the more 'traditional' metrics like SSIM or NRMSE. However, LPIPS is trained primarily on images of macroscopic objects, and it is questionable if it can be applied well to microscopic data, which has yet to be thoroughly explored. This may continue to be an issue with other future metrics, which are developed primarily by computer vision groups for use on macroscopic images. Hence, a future focus of the bioimaging community using DL-tools for image reconstruction should be the development of quality metrics which can be used more safely and intuitively on bioimages than those in use today. This argument could even be extended to other aspects of DL-tools in bioimaging, such as the use of loss functions from methods developed for computer vision tasks on 'natural images'. It is conceivable that accommodating features typical of bioimages, such as clear background-foreground distinctions or images with few colours, into all aspects of the DL pipeline, including QC metrics, could yield better results than methods adapted from 'natural image'-based computer vision tasks. Another common datatype in bioimage data which has hardly been addressed in terms of quality metric design is the analysis of 3D data. In this project, this was done by performing QC on the slices of a stack and then averaging the scores to estimate the result for the entire stack. Although it can be useful for the user to evaluate their models in this way, it may miss important aspects of model performance in the reconstruction of volumetric information. Again, the metrics which are borrowed from computer vision tasks which are primarily concerned with 2D data, may need to be improved for better QC to be performed for DL-models working on 3D data which is very common in bioimaging.

II. 7. Results Chapter 2 - Summary

To summarise, the QC section in the ZeroCostDL4Mic notebooks is suitable to give users useful insight into the quality of the models trained in the notebooks. It achieves this by firstly, integrating it easily into the workflow and providing a rich set of information which allows parameter adjustments to be made to fine-tune the models. This is crucial for novice users to explore how model behaviour changes and how it affects the predictions on unseen data. The analysis which gives qualitative and quantitative feedback to the user provides some transparency to DL-methods which can quickly produce undesirable predictions or appealing ones with errors that are difficult to detect by eye. Therefore, it fulfils its purpose in the context of this project in enabling a new type of user to access DL tools and in providing an avenue to improve the reliability of DL tools, thus addressing two of the previously outlined problems for DL adoption, the 'knowledge' and ‘reliability’ problems.

While attempting to show that the models trained here can be evaluated using the QC metrics, it was incidentally observed that the metrics used to evaluate different DL-tasks differ strongly in their informational value to the user, with metrics for classification and segmentation tasks providing relatively good predictors of model performance as estimated by visual inspection of model outputs, and metrics for image-to-image tasks requiring careful additional analysis of the output images to give insight into true model performance. The difficulty of assessing how well DL-models perform in certain tasks calls for the community to set standards for model evaluation and to focus on developing new standards and metrics for a class of algorithms which is likely to be widely used for imaging tasks in the future.

II. 8. Methods

Metrics

Mean-squared error (MSE)

The MSE between a ground-truth image (GT) and a corresponding predicted image (P) is defined as:

$$MSE = \frac{1}{N}\sum_{i,j}^{N}\left(GT_{i,j} - P_{i,j}\right)^2$$

Where i and j represent coordinates of pixels in the ground-truth and predicted images and N represents the total number of pixels in the image. In the notebook this error is further simplified by taking the root of this value which becomes the root-mean-squared error (RMSE):

$$RMSE = \sqrt{\frac{1}{N}\sum_{i,j}^{N}\left(GT_{i,j} - P_{i,j}\right)^2}$$

Since the images predicted by the trained model may not correspond to the same dynamic range as the ground-truth image, the predicted image is normalised with respect to the ground-truth by percentile normalisation as follows:

$$P_{ij}^{norm} = \frac{P_{ij} - P_{99.9}}{P_{99.9} - P_{0.1}}$$

where P_{ij}^{norm} represents the normalised intensity value at pixel (i,j), P_{ij} the intensity to be rescaled, $P_{99.9}$ the value of the pixel which lies in the 99.9th percentile of pixel values in the image and $P_{0.1}$ the pixel value of the pixel in the 0.1th percentile of the pixel values in the image. This percentile-based normalisation (instead of using minimal and maximal pixel values) is aimed at preventing the influence of dead or hot pixels which are common in microscopy images and may distort the useful dynamic range of the image upon normalisation.

The linear regression normalisation is done the same way as in CARE, based on least square minimisation, defined as:

$$(\alpha_o, \beta_o) = argmin_{\alpha,\beta} \sum_{i,j}^{N} \left(GT_{ij} - (\alpha P_{ij} + \beta)\right)^2$$

where GT_{ij} and P_{ij} are the respective pixels in the ground truth and prediction and α and β the parameters which rescale the prediction to the dynamic range of the ground-truth.

The normalised image is then calculated as:

$$P^{norm} = \alpha_o P + \beta_o$$

Using this normalisation protocol, we can calculate the final error metric, the normalised root-mean-squared error (NRMSE):

$$NRMSE = \sqrt{\frac{1}{N}\sum_{i,j}^{N}\left(GT_{i,j} - P_{i,j}^{norm}\right)^2}$$

The NRMSE metric gives a positive value between 0 and 1, where small values correspond to a smaller detected error between the images. Since this metric is calculated in a pixel-dependent manner, it can be valuable to visualise the calculated errors per pixel to produce an error map. This can be done simply by adapting the formula for the RMSE to evaluate each pixel and plot the value in an array, giving a root squared error (RSE):

$$RSE_{ij} = \sqrt{\left(GT_{ij} - P_{ij}^{norm}\right)^2}$$

Peak-signal-to-noise ratio (PSNR)

PSNR measures the same quantity as the MSE, the pixel-wise error between a ground-truth image and a prediction. Intuitively, the PSNR is the ratio the between the maximal (peak) expected signal in the image and the error (MSE) between prediction and ground-truth, meaning the PSNR becomes higher the lower the MSE becomes compared to the maximal dynamic range of the signal. Given the potentially large dynamic ranges at certain bit-depths or very low MSE, PSNR is usually given in decibels, i.e. at a logarithmic scale. It is thus defined as follows:

$$PSNR = 20\ \log_{10} \frac{\max\ (GT)}{\sqrt{MSE(GT,P)}}$$

where *max(GT)* represents the maximum pixel intensity of the ground-truth, which is usually defined by the bit-depth of the image, i.e. 255 for an 8-bit image. *MSE(GT,P)* represents the mean squared error between the GT and the predicted image, defined as above.

Structural Similarity (SSIM)

The SSIM between two images X and Y is calculated as introduced by Wang et al.:

$$SSIM(X,Y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

where μ_x and μ_y refer to the mean pixel intensity, σ_x^2 and σ_y^2 to the variance of pixel intensities and σ_{xy} to the covariance of the pixel intensities between the images. C_1 and C_2 are introduced to avoid instability for small denominators and are determined using the bit-depth L of the images as follows:

$$C_1 = (K_1L)^2$$
$$C_2 = (K_2L)^2$$

With K_1= 0.01 and K_2= 0.03 as suggested in one of the original SSIM publications[193]. In the notebooks, this is implemented using the skimage metrics structural_similarity package.

To ensure that SSIM can be calculated without effects related to different dynamic ranges of predictions and ground-truth images, before QC the images are normalised to values between 0 and 1, first by percentile normalisation on all data (source, target, and prediction):

$$I_{ij}^{norm} = \frac{I_{ij} - I_{99.9}}{I_{99.9} - I_{0.1}}$$

where I_{ij}^{norm} is the normalised intensity of the pixel with coordinates i, j. I_{ij} is the intensity to be rescaled at the pixel coordinates i, j. $I_{99.9}$ and $I_{0.1}$ are the intensities of the pixels lying in the 99.9th and 0.1th percentiles of intensities in the image, respectively.

A further normalisation is performed on source and predicted images via linear regression compared to the target as proposed by the authors of CARE[113] as:

$$(\alpha_o, \beta_o) = argmin_{\alpha,\beta} \sum_{i,j} \left(GT_{ij} - \left(\alpha I_{ij} + \beta\right)\right)^2$$

where GT_{ij} and I_{ij} are the respective pixel coordinates in the ground truth and predicted images and α and β the parameters which rescale the prediction to the dynamic range of the ground-truth.

These parameters then rescale/normalise the image *I* as:

$$I^{norm} = \alpha_o I + \beta_o$$

After rescaling, any pixels with intensities above 1 and below 0 are thresholded to 1 and 0, respectively.

The SSIM map is calculated on a local window around the pixel of interest. The window size is set to 11x11 pixels and a Gaussian weighting function of 1.5 pixels standard deviation. The global SSIM metric is calculated by averaging the SSIM value over the entire SSIM map.

Intersection over Union (IoU)

IoU is calculated by dividing the number of pixels shared between predicted and ground-truth masks by the total number of pixels in the union of the two masks:

$$IoU = \frac{I \cap GT}{I \cup GT}$$

where I represents the predicted image and GT the ground-truth image.

Average precision (AP)

To calculate AP in the QC section, three metrics in the model's output are used: bounding box coordinates, predicted class labels, and the confidence, i.e. the likelihood the model estimates for its predicted class label to be correct. All object detections of a class are ranked and listed by confidence from highest to lowest. This list is used to calculate the precision and the recall of the model per object:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

True positives were defined as detections with correct class labels and predicted bounding boxes with a minimum IoU to the ground truth bounding boxes of 0.3.

As more objects on the list are evaluated, the precision and recall values change, with both values logged for each object. The recall and precision values are then plotted against each other beginning at the top of the list, meaning the objects with the lowest confidence are found furthest to the right on the p-r plots.

In ZeroCostDL4Mic QC, the AP is calculated as proposed in the PASCAL VOC challenge by interpolating precision scores. This means that precision at recall r is equal to the maximum precision for any $r' \geq r$, leading to a step-like shape of the curve. This reduces the effect of individual detections in the data on the AP.

The mean average precision (mAP) is calculated as the average *AP* of all classes (*n*) the model predicts:

$$mAP \ = \frac{1}{n}\sum_{i}^{n} AP_i$$

where *i* represents an individual class.

F1 score

The F1 score is the harmonic mean of precision and recall:

$$F1 \ = 2 \ \frac{Precision\ x\ Recall}{(Precision + Recall)}$$

As the highest values for precision and recall are 1 the best possible score for F1, i.e. when the false positive and false negative rates over the entire test dataset are zero, is also 1.

Panoptic quality

Panoptic quality is a metric that was developed to score tasks which perform semantic segmentation and instance segmentation simultaneously. This means that objects are not only distinguished by class but also from one another within each class (“stuff” and “things”)[205]. The metric is therefore designed to account for both, the precision of detections as well as the quality of the segmentations for each object. It is calculated as:

$$PQ = \frac{\sum_{p,gt \, \epsilon \, TP} IoU(p,gt)}{|TP| + \frac{1}{2}|FP| + \frac{1}{2}|FN|}$$

Where *p* and *gt* are the predicted and ground-truth segmentations, respectively and other symbols are defined as above. The panoptic quality can also be interpreted as a combination of the summed IoU score over the sum of TP and the F1 score of the detected objects, by multiplying by $\frac{|TP|}{|TP|}$:

$$PQ = \frac{\sum_{p,gt \, \epsilon \, TP} IoU(p,gt)}{|TP|} \; x \; \frac{|TP|}{|TP| + \frac{1}{2}|FP| + \frac{1}{2}|FN|}$$

PQ ranges from 0 (worst) to 1 (best), and gives an intuitive understanding of the model's overall performance in detection and prediction.

Analysis of QC results

The values for the metrics of the CARE and fnet experiments were extracted from the QC_results.csv files using the csv Python module and then plotted using the seaborn Python library.

Notebooks

The ZeroCostDL4Mic notebooks used in this chapter were the StarDist2D (v1.13), YOLOv2 (v1.13), CARE2D (v1.13) and fnet_3D (v1.13) notebooks, which can be found and accessed through the ZeroCostDL4Mic project GitHub page: https://github.com/HenriquesLab/ZeroCostDL4Mic/wiki

Datasets

YOLOv2

The dataset used here is the same as described in chapter 1. To change the number of classes in the dataset, the original PASCAL VOC files created for the ZeroCostDL4Mic publication[198], were copied and edited using a Python script to create several new datasets with annotations for only the specified number of classes.

StarDist 2D

This dataset is part of the ZeroCostDL4Mic datasets acquired by Guillaume Jacquemet and Johanna Jukkala and its creation is described in the methods of the original publicaton[134,199] Briefly, the nuclei of DCIS.COM LifeAct-RFP cells were labelled using a SiR-Hoechst label and acquired for 14 hours on an inverted Zeiss Axio Observer Z1 spinning disk confocal microscope with a 20x (0.8NA) dry objective. A set of images was then hand-labelled in Fiji using the ROI manager and the LOCI plugin as described in more detail on the project's wiki page: https://github.com/HenriquesLab/ZeroCostDL4Mic/wiki/Stardist

CARE 2D

LifeAct

This dataset is part of the ZeroCostDL4Mic project and was acquired by Guillaume Jacquemet. Its creation is also described in the original publication[134,197]. Briefly, DCIS.COM lifeact-RFP cells were grown to confluence, washed in PBS, and labeled with phalloidin 488 at 4 degrees overnight. Images were acquired on a DeltaVision OMX v4 microscope (GE Healthcare Life Sciences) microscope with a ×60 Plan-Apochromat objective (1.42 NA). To acquire high SNR images, cells were imaged for the phalloidin label using the SIM modality, with 5 phases and three rotations. The low SNR images were acquired in widefield settings and for the endogenous LifeAct-RFP label. The dataset consists of maximum intensity projections of the images.

β-tubulin and TOM20

These datasets were kindly provided by Christoph Spahn, at Goethe University, Frankfurt. This protocol is derived from him. Briefly, HeLa cells were grown on fibronectin and labelled with an Alexa Fluor 594 antibody for the mitochondrial outer membrane domain TOM20, and for β-tubulin using AbberiorSTAR 635SP. The cells were acquired on a Leica SP8 confocal microscope with a 100x (1.4NA) oil immersion objective (HC PL APO CS2), using a photomultiplier tube for capture. High and low SNR images were acquired using 8x an 1x line averaging during acquisition and 1% and 0.1% laser power, respectively.

Label-free prediction – fnet 3D

This dataset is part of the ZeroCostDL4Mic project and was acquired by Christoph Spahn. Its creation is described in detail in the publication[134,201]. Briefly, HeLa ATCC cells were fixed and labelled for the mitochondrial membrane domain TOM20 with a rabbit anti-TOM20 primary antibody (sc-11415, Santa Cruz, USA) and a donkey-anti-rabbit-secondary antibody with a conjugated AlexaFluor 594 fluorescent label. The cells were washed in PBS and images were acquired on a Leica SP8 confocal microscope with 63x (1.40NA) oil immersion objective. 25 stacks with dimensions of 1024x1024x32 were acquired and split into smaller FOVs of dimensions 512x512x32 to create the training dataset containing 92 images for training and 8 for quality control.

IV. Results Chapter 3 – A ZeroCostDL4Mic showcase for an image analysis challenge

IV. 1. Establishing unmixing as an imaging task for DL

In the previous two chapters, I outlined the creation of the ZeroCostDL4Mic platform, and how the implemented quality control (QC) methods help in confirming the implemented DL-tools work as expected in the Colab environment and that the platform allows model fine-tuning for the intended tasks. Having created a platform which improves ease-of-access to DL-tools and showing that the outputs can be quality-tested, in this chapter I will demonstrate how the platform can be used for a custom bioimage analysis challenge. This final step is needed to show that the tool is indeed suited to and versatile enough to allow users to develop new and custom tasks that employ DL for their research.

I aim to achieve this by showing the use of DL methods implemented in ZeroCostDL4Mic for the unmixing of different biological structures from the same imaging channel, which is related to the key challenges in microscopy, outlined in the introduction of this work. I show how the ZeroCostDL4Mic platform is suited to set up a study to find a suitable method for this challenge, and how the results can even aid the user to explore datasets in ways which may be difficult without the DL and QC methods implemented in this project.

To understand why the above challenge was chosen for this chapter, we can first define what channel unmixing entails and where this challenge may be encountered. Biological specimens are often imaged using two or more channels, usually with fluorescent labels tagged to a specific subcellular structure of interest, through endogenous expression or targeted labels, such as antibodies or dyes. Imaging cells in multiple channels is done for various purposes. One example is the study of the interaction between different structures, for example protein complexes, which may or may not colocalize, giving insight into their functional relationships[207]. Other reasons may be more pragmatic. For example, for studies involving cell tracking a nuclear label is often employed in addition to other labels of interest, as each cell usually has only one nucleus and because nuclei can be easily detected with computational methods[45,208]. Multichannel imaging studies are usually performed by using multiple fluorophores that have emission spectra with clearly separable peaks, meaning that there is little overlap between emission spectra. This reduces the risk of the so-called bleed-through that occurs when different biological structures appear in the same channel as the structure of interest and is caused when the emissions of different fluorescent labels are detected in the same detector. If unidentified, bleed-through can result in misleading images of biological structure and erroneous conclusions about localisation or function. However, if the emission spectra do not overlap significantly, they can be imaged simultaneously for example using wavelength-specific filters. Alternatively, channels can be imaged separately. The latter approach means a short time delay between channels, which can hinder some fast live-imaging experiments, as this could lead to channels misaligning in the resulting timelapses.

Multichannel imaging is often assisted by computational methods, which can aid in separating channels with overlapping emission spectra and is thus often used to compensate for bleed-through. The most popular method is linear spectral unmixing[209]. Here, the total image signal is assumed to be a linear combination of each fluorophore’s emissions. When the signal and emission spectra for each fluorophore are known, the contribution of each can be estimated by solving a relatively simple differential equation[209]. For linear unmixing to work, the emission spectra of all fluorophores need to be known and the number of detection channels must be at least equal to the number of fluorophores in the sample. This means the number of channels for which unmixing is done is generally limited to the number of detection channels in the microscope. As explained above, this form of unmixing is mostly used to reduce the effects of bleed-through which tend to be relatively small and significant overlap between labels can make linear unmixing much more difficult.

More recently, some groups used a DL approach to perform spectral unmixing. McRae et al. show that an unsupervised model can learn to unmix overlapping channels in an image with significant overlaps between channels[210]. The paper provides an ImageJ plugin which can be relatively easily employed for custom datasets. As in linear unmixing, the primary purpose of the tool is to reduce bleed-through effects in imaging tasks.

Though useful for many users, this misses the potential that DL-powered unmixing could have if exploited in more extreme cases, for example, if channels are purposely acquired in the same detector and the same colour. If DL could be trained for tasks where all or some labels are acquired in the same channel, i.e. with significant spectral overlap, it would allow multiple different biological structures to be imaged using only one fluorescent label or multiple labels with overlapping excitation spectra. This has the potential to reduce the light dose that a biological sample is subject to per acquired frame, thus reducing the risk of damaging the specimen through phototoxicity. This approach could also be used beyond the pure reduction of phototoxicity as it could be exploited to image additional biological structures, which would otherwise be difficult to acquire simultaneously. For instance, if a cell’s nucleus and mitochondrial network can be distinguished via DL, there would be no requirement to label them using different fluorophores, and instead an additional structure could be acquired in the channel previously occupied for the imaging of mitochondria or nucleus. Depending on how many channels can be successfully ‘unmixed’ by a DL algorithm in this way, the number of structures which could be labelled in a single imaging experiment could quickly be extended with applications e.g. for drug-screens.

Another advantage is that unless the respective microscope has a multiple bandpass filter which allows several different colour channels to be imaged simultaneously, DL-enabled ‘unmixing’ could reduce the time required to capture a multichannel image, as neither light-source nor filter would need to be changed during frame acquisition. This could improve the speed of timelapse acquisitions and increase temporal resolution and potentially make multichannel imaging more feasible on microscopes without band-pass filters.

Given these potential advantages of a computational approach to distinguish biological structures and since there is limited literature on this challenge in a DL context, this acts as an example of a task a bioimage analyst or microscopist may encounter and for which DL may be an avenue worth exploring. In this chapter, I show how the tools implemented in ZeroCostDL4Mic provide such an avenue and allows users to test and evaluate DL tools for such novel imaging challenges.

To demonstrate the feasibility of the approach, I created a custom dataset with three channels from a publicly available resource and tested three methods available in the ZeroCostDL4Mic platform for the above-defined unmixing task. The methods were then evaluated using the QC section for image-to-image comparison as described in the previous chapter. For some of the methods, I show how fine-tuning can improve performance in this task. Beyond the performance of the models themselves, there were other key aspects which are important in determining whether the platform created in this project succeeds in facilitating this task. A key question for users of the platform is whether it is feasible to train a model to the highest possible performance without timeouts and whether small performance improvements justify significantly longer training times or the risk of frequent timeouts. For the ZeroCostDL4Mic platform or its tools implemented to facilitate democratisation, as is the aim of this project, notebooks should be able to allow users to perform these tasks even if the resources on Colab may be limited compared to custom DL workstations.

In the final section of this chapter, I discuss how the above results help to understand the applicability of the ZeroCostDL4Mic platform for the bioimaging community and the challenges that remain. Ultimately, these considerations will be used to assess in how much the tool contributes to the democratisation of DL for microscopy, which this project attempted to achieve.

IV. 2. Creating a ZeroCostDL4Mic dataset for channel unmixing

To demonstrate that channel unmixing is achievable using the DL methods on the ZeroCostDL4Mic platform, a publicly available dataset was used which was originally published for the label-free prediction method (fnet)[114] and used three fluorescently labelled structures, namely nucleus, mitochondria and membrane to be used in the unmixing experiment. This dataset was chosen because its size was suitable for the ZeroCostDL4Mic platform and allowed testing of both 3D and 2D networks and because it was difficult to acquire such images during the pandemic. The structures of interest that were chosen from this dataset represent three commonly labelled structures in bioimaging, thus, structures that are likely targets for a task that requires channel unmixing. The labels were chosen because they occupy different regions of the cells, which would be expected to facilitate the task and reduce the risk of failure.

The dataset contained 75 3D stacks which were divided into a dataset of 68 stacks for training and 7 for quality control. The stacks were cropped to a format of 512x512x32 to ensure that each image and the full dataset could be loaded into the RAM of any ZeroCostDL4Mic notebook. The images were normalised as 16-bit images. To create the combined channels which the DL-methods were to ‘unmix’ the pixel intensities were combined by simple pixel-wise addition, with values above the value of 65335 clipped to this value. To create 2D datasets, each stack was split into individual slices, giving a training dataset of 2176 image pairs for training and 224 for QC.

The combination of channels in this way simulates the case where multiple structures are acquired using fluorescent labels with the same or nearly the same emission peaks. This means that the labels show significant overlap in some areas of the images (see Fig. 41) with all labels contributing equally to the whole signal.

IV. 2. 1. Parameter choices

First, all networks were trained on the task of recovering individual labels from the merged labels. As the different neural networks underlying the methods and the way the data is handled in each method differ significantly, some additional considerations were necessary before comparing the models for this task. The main consideration when choosing parameters for the models was how long it would take to train the models, since the datasets were large in relation to the datasets used in the previous chapters. First, a small patch size of 128x128 pixels was chosen for the 2D models and 128x128x32 pixels for the 3D models to minimise the memory footprint in the notebook environments. Furthermore, although performance improvements could be expected from the results of the previous chapter and those seen in the publication, the datasets were not augmented for training, initially. All models were trained on a patch size of 128x128 pixels (for 2D methods) and 128x128x32 (for 3D methods). Since fnet patches the dataset automatically, and CARE does not, for CARE the number of patches was set to 16 for all experiments, as this should cover the full field of view of each image, and which should be the same as in fnet. The validation split was 10% for both the CARE and fnet models. For the pix2pix models these considerations were not made as these models do not use a validation split. Furthermore, the pix2pix models do not accept a patch size smaller than 256x256, which means that the same patch size could not be used for this model. More generally, the pix2pix method is an outlier as it requires images in .png format which necessarily means training on a lower bit-depth (8-bit) than the other models which use 16-bit .tifs. Hence, the performance of the pix2pix models needs to be considered somewhat separately from the other models.

To determine the settings for epochs or steps, the main considerations are to avoid overfitting in CARE, which was a problem in the previous chapter, and to determine a suitable number of steps (batches for the fnet model). For the former, training was initially performed with 100 epochs for the experiment. However, from the first trial, models began overfitting significantly earlier, which meant that better performances of the models would likely be reached earlier, also allowing faster completion of the training for each method. Thus, the number of epochs was reduced to 50 for the 2D models and 30 for the 3D models which led to models less prone to overfitting on the datasets.

For fnet, assuming 61 stacks for training and 7 for validation, the number of steps was calculated as in Chapter 2, for the 2D notebook:

(61 images x 32 slices x 16 patches per image / batch size 64) x 100 = 48800 steps

And for the fnet 3D notebook:

(61 images x 16 patches per image / batch size 8) x 100 = 12200 steps.

These batch sizes were chosen to maximise the RAM capacity of the notebooks. For CARE the batch size parameter is not as relevant as it simply adjusts the number of steps per epoch. Hence, for CARE batch size was simply kept at the default value of 16.

The parameter settings for the initial experiments were kept as similar as possible in the initial experiment, as one result of interest was which method would perform best on the unmixing tasks. Hence, it was necessary to eliminate sources which could explain performance differences. However, since the architectures differ, some models may have been favoured by this initial choice of parameters. After the initial experiment with the above settings, the performance of each method was improved by fine-tuning the parameters of the notebooks. The results of that experiment are shown in the second part of the chapter.

IV. 2. 2. Visual analysis and example data

In the first experiment, models were trained to predict one label, either the nuclear, membrane, or mitochondrial label, from a composite image of all three. This task requires the network to learn the differences between the structure of interest and those to be filtered from the image. The performance of the models can be measured using different metrics. The first are the quality metrics which were introduced in the previous chapter, and the second is the assessment of the time it needs the models to reach the performance. The latter assessment is unique to Colab because of its inherent limitations. If the unmixing task or similar tasks are to be performed on the platform, it is of interest for example if the training can be performed within Colab's runtime limitations and with the allocated GPUs. To facilitate the analysis of the results, the models' performance is assessed for each label separately before comparing the models' overall performance. The analysis of the results follows the guidelines of the notebooks, with visual analysis followed by quantitative analysis. This is also based on the results from the previous chapter which suggests that the metrics across the dataset are more reliably interpreted when combined robust visual controls of example predictions and error maps. The losses of the models are not shown in this section as they will be of more interest when finetuning the parameters and will therefore be shown in the section below (*Finetuning*) for some of the models trained here.

IV. 2. 2. a. Unmixing 3 labels to nucleus

The first impression from inspection of the predictions is that all models perform well, visually, at reconstructing the nuclear signal from the composite image, with both large-scale and small-scale structures well-recovered from the input. Differences between predictions are difficult to see. One notable difference is that the 3D methods appear to recover smaller foci of nuclear label better than 2D models which mainly omit these labels or show them as vague blurs.

Differences between the models are more apparent on the error maps. The U-net-based networks, CARE 2D and 3D and fnet 2D and 3D, are better at predicting the nuclear signal from the composite than the pix2pix model, which is based on a GAN architecture and training. However, this performance mismatch appears to be mainly due to the slightly worse performance of pix2pix in reconstructing the background signal than the nuclear structure, which appears to be very similar to the ground truth in both pix2pix and the other models (Fig. 41). Next, the error maps confirm the initial visual impression that in both CARE and fnet, the 3D methods perform better than the equivalent 2D methods. According to the error maps which show fewer misrepresented areas and are overall brighter in the SSIM map, suggesting higher similarity, and darker in the RSE maps, suggesting fewer pixel-wise errors. Overall, the performance of the fnet models appears slightly superior to the CARE models, with the best prediction achieved by the fnet 3D model. While CARE 3D is superior to fnet 2D, it appears to predict the outlines of the nuclei less accurately than fnet 3D which is most apparent in the SSIM error map which shows a dark (dissimilar) edge around the nuclei in the CARE 3D case.

As a second assessment criterion, it can be asked how easily it was to train the models for this task in Colab. Here, the CARE models are by far the models which required the least training. On the fastest allocated GPU (Tesla P100C), CARE 2D models trained in 45 minutes for this task, with the fnet 2D models trained in 3 hours 45 minutes (48800 steps, batch size 64). The 3D methods required more time for training, with CARE requiring 58 minutes (batch size 8, 30 epochs) and fnet 8 hours and 46 minutes (12200 steps, batch size 8). The pix2pix model was trained in 1 hour and 34 minutes. The QC for the methods was comparably fast for the CARE and fnet methods, requiring less than 5 minutes for the analysis of the full test datasets. In contrast to the other methods, pix2pix takes longer for this step as it is not as memory efficient in QC. This is because its control in the ZeroCostDL4Mic notebook requires every 10th checkpoint from the models' training to be assessed for each test image which requires significantly more time than the other methods. It is also not as memory efficient and requires significantly more drive space. Indeed, the pix2pix QC could quickly exceed the Google drive limit of a basic account, and for further models to be trained required additional drive memory to be purchased.

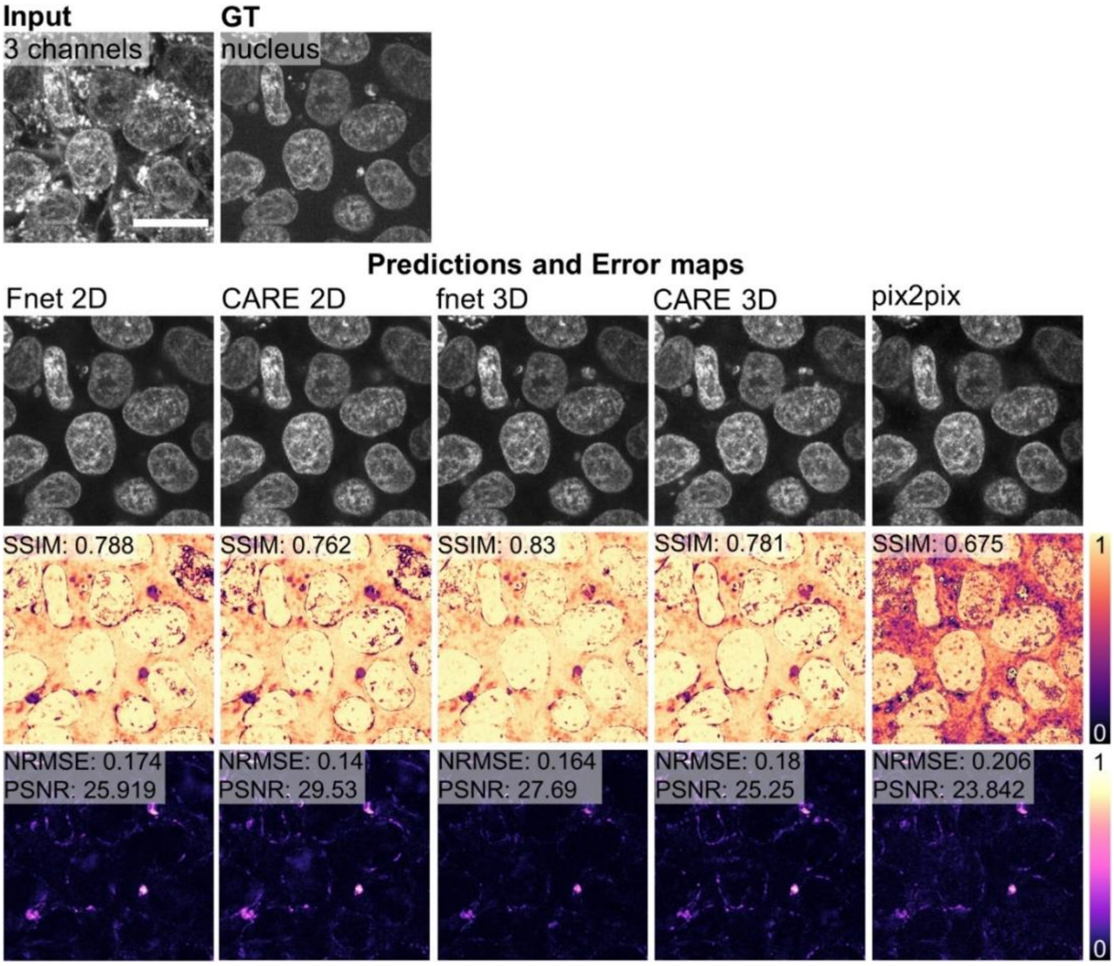


Figure 41 – Unmixing the nuclear channel from a three-channel composite *– Top: Example of a test image pair, with the input image containing three channels (membrane, nucleus, mitochondria) and the output image containing only the nuclear channel. Lower panel: Top: Predictions from different models trained to filter nuclei from composite images. Middle: Error maps, showing the pixel-wise SSIM between the predictions in the above row and the target shown in the top panel. Bottom: Error maps, showing the pixel-wise RSE between predictions in the above row and the target shown in the top panel. The metrics in shown in the error maps represent the value for the shown example image only. (scale: 20µm)*

IV. 2. 2. b. Unmixing 3 labels to membrane

The reconstruction of the membrane label is largely successful. All models perform well in identifying the small foci in the membrane channel which appear throughout the image, even in relatively crowded areas in the composite image (Fig. 42 – top row). However, some models also incorrectly predict foci where there are none in the target (see bottom left corner of the example image). The membrane itself is also recovered in all images, however, to different degrees. In the CARE models and pix2pix, the membrane is predicted with relatively ‘fuzzy’ edges and contains some discontinuities. This is particularly apparent in the predictions of the CARE 3D model, where many of the membrane structures appear to merge with the background. In contrast to the recovery of the nuclear signal, the CARE 3D model performs worse than the 2D model on this image, which does not predict as many artefacts between the membrane structures. The fnet models perform comparatively better. Again, this is more apparent in the error maps, particularly the SSIM maps (Fig. 42 – middle row). The 2D fnet model predicts fewer artefacts than the 2D CARE model and achieves much better contrast between the edges of the membrane structure and the background. The fnet 3D model is clearly the model achieving the highest similarity with the ground-truth, showing fewer of the dark outlines around the membrane filaments and predicting fewer ‘false’ foci than the other models. Overall, performance appears to be lower than that for the recovery of the nuclear label, according to the quality metrics. This may suggest that this task is more challenging to learn for all methods.

The method training time was identical to the previous task, as the images had the same file size and bit depth.

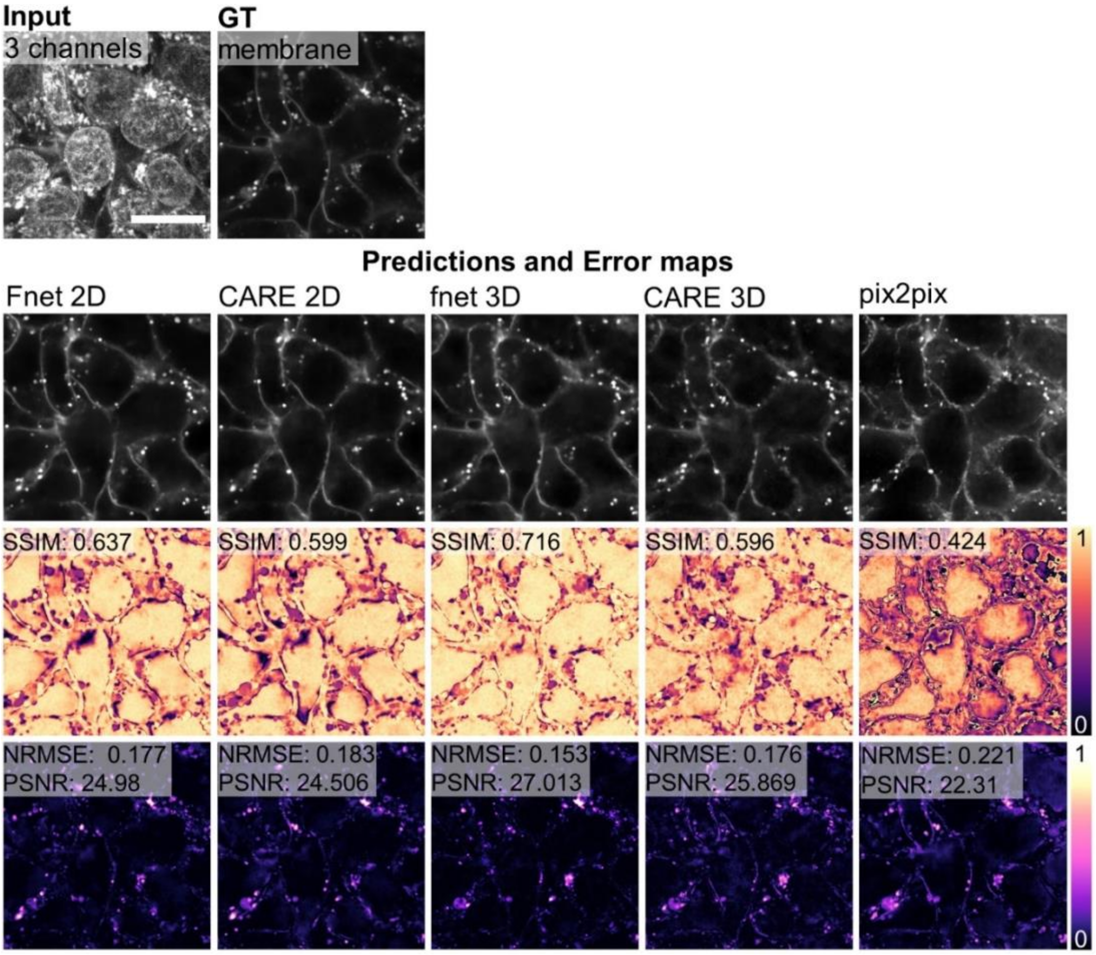


Figure 42 – Unmixing the membrane channel from a three-channel composite – *Panels as in Fig. 41, with the target membrane. (scale: 20µm)*

IV. 2. 2. c. Unmixing 3 labels to mitochondria

The mitochondrial channel appears to be the most difficult to recover from the composite image. Although all models predict the general localization of the mitochondrial network correctly from the composite image, much of the small-scale detail in the structures is not correctly predicted (Fig. 43 – top row). However, these differences are difficult to identify from the predictions. Although the error maps suggest that the 3D models have a higher similarity to and smaller discrepancies with the ground-truth than in the 2D models, this appears primarily through the better recovery of the background rather than the signal of the mitochondrial label (Fig. 43 middle row). The SSIM maps show many errors within the predicted structures, which is unlike the recovery task for the previously shown labels where the errors were primarily located outside the predicted structures, i.e. in the background. Both CARE models predict artefacts in the form of dim signal between the mitochondrial structures, reducing the contrast between mitochondria and background. In the fnet models, this effect is less pronounced but still notable. Furthermore, the thinnest and smallest structures in the ground-truth are missed partly in the 3D models and nearly totally in the 2D models' predictions. The pix2pix model's prediction is revealed in the error maps to have significant structural artefacts. This may be explained by the manner in which the pix2pix model was chosen. The best model checkpoint for this pix2pix model according to the QC was the tenth checkpoint. Thus, the observed artefacts may be the result of the early training stage from which the model was picked and may disappear in later training points. This again highlights that the visual inspection of the predictions and error maps implemented in the ZeroCostDL4Mic platform is crucial in interpreting model performance.

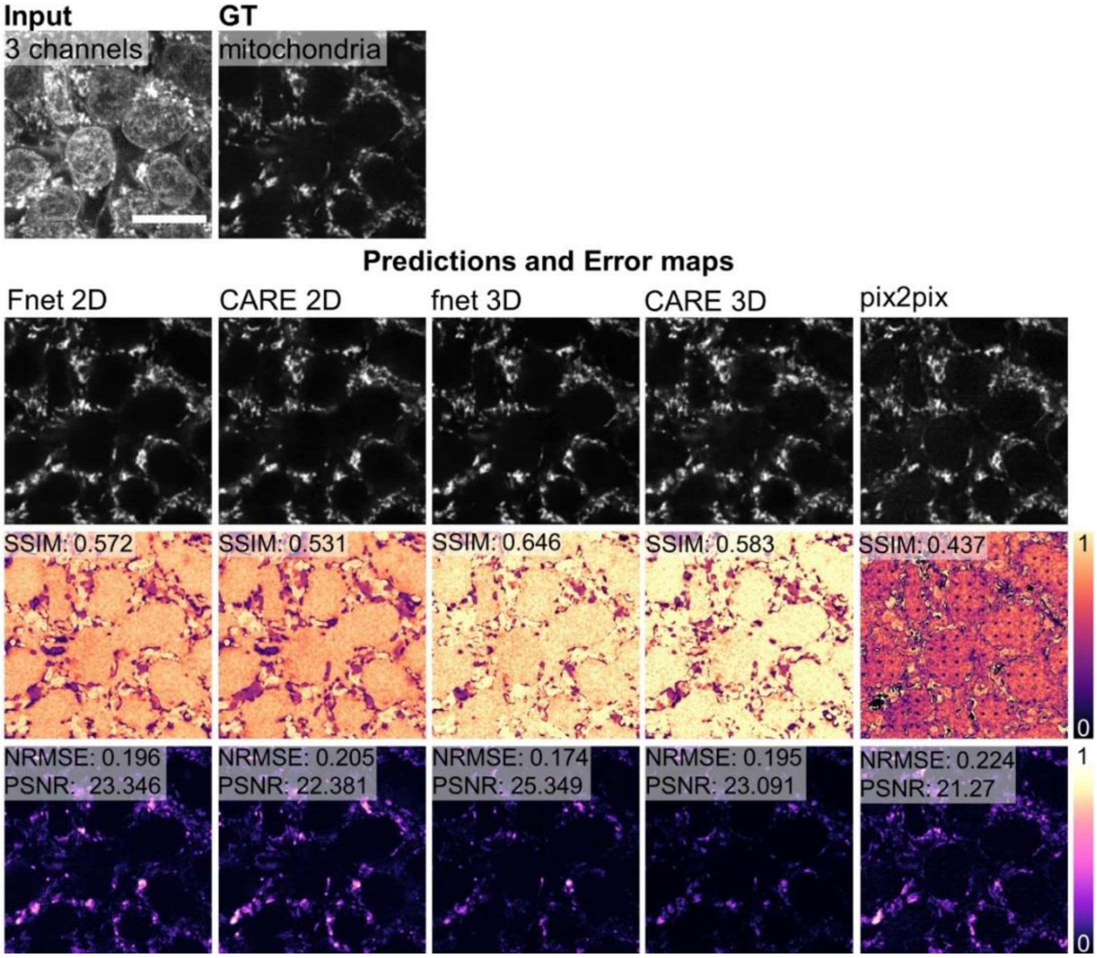


Figure 43 – Unmixing the mitochondria channel from a three-channel composite – *Panels as in Fig. 41, with the target being mitochondria. (scale: 20µm)*

IV. 2. 2. d. Unmixing membrane + nucleus

Initial experiments in 'unmixing' individual channels from a composite of three showed that DL tools in the ZeroCostDL4Mic project could be used for this task, with some limitations, as seen, e.g, for the prediction of mitochondrial signal. As multiple channels overlapping one another can be a confounding factor for models learning the identity of the structures to remove from the image, in the next set of experiments, I tested if performance could be improved by removing an additional channel from the input images. This was done to identify which channels that overlap may be more difficult to distinguish in the composite image than others. For the mitochondrial label, it was of interest to know if the removal of an additional channel could improve the performance of unmixing in any of the models. To perform these experiments, three additional datasets were constructed from the above data which contained the following label-pairs: Nucleus-Membrane, Mitochondria-Nucleus, and Membrane-Mitochondria. For each experiment, two models of each method were trained. One of the 2-channel datasets was used as an input channel and one of the constituent channels was used as target for one of the two models, respectively. This led to a total of 6 models per method for this experiment. Below I go through the results for each channel pairing and the models' predictions.

In the first dataset, the mitochondrial channel was removed from the composite image. As expected, all models improve in their unmixing performance once the mitochondrial label is removed (Fig. 44). The membrane channel predictions appear to have higher contrast and fewer artefact foci, especially in the CARE models, when compared to the previous predictions of this channel (Fig. 42 – top panel). The CARE 3D model predicts the membrane structure with more contrast against the background than in the previous experiment and with fewer blurred edges around the membranes. The performance improvement is most visible in the error maps which show fewer dark areas in the SSIM maps and fewer bright spots in the RSE maps (Fig. 44 – top panel: middle and bottom rows). It can also be observed that the fnet models outperform the CARE models and their own predictions from the three-channel unmixing. The pix2pix model improves over the previous performance by the largest margin in terms of SSIM and RSE but remains the model predicting images with the lowest similarity to the ground-truth. As in the previous experiment the fnet 3D model clearly outperforms the 2D version while the CARE 2D and 3D methods do not appear to differ much, with the 2D method potentially being even better at this task, at least on the example image.

In the prediction of the nuclear signal, a significant improvement is clear for all models from the first inspection of the data (Fig. 44 – lower panel). The similarity between prediction and target is high for all models, with an improvement specifically apparent for the CARE 2D and pix2pix models compared to the previous experiment (Fig. 41). All models in this experiment now predict the smaller nuclear labels located between nuclei which previously was mainly achieved by the fnet 3D model. The error maps also indicate that the internal structure of the nuclei is recovered more closely to the ground-truth than in the previous experiment, with fewer errors being highlighted inside the nuclei (Fig. 44 – lower panel: middle row).

Another indicator for the good performance of the models is that both background and foreground structures have an almost identical appearance in the SSIM map. This means that the models do not boost their performance by learning primarily to recover the background, as was seen in some images in Chapter 2. This also suggests that the scores of the metrics more closely relate to the performance in the desired task, unmixing, than in denoising.

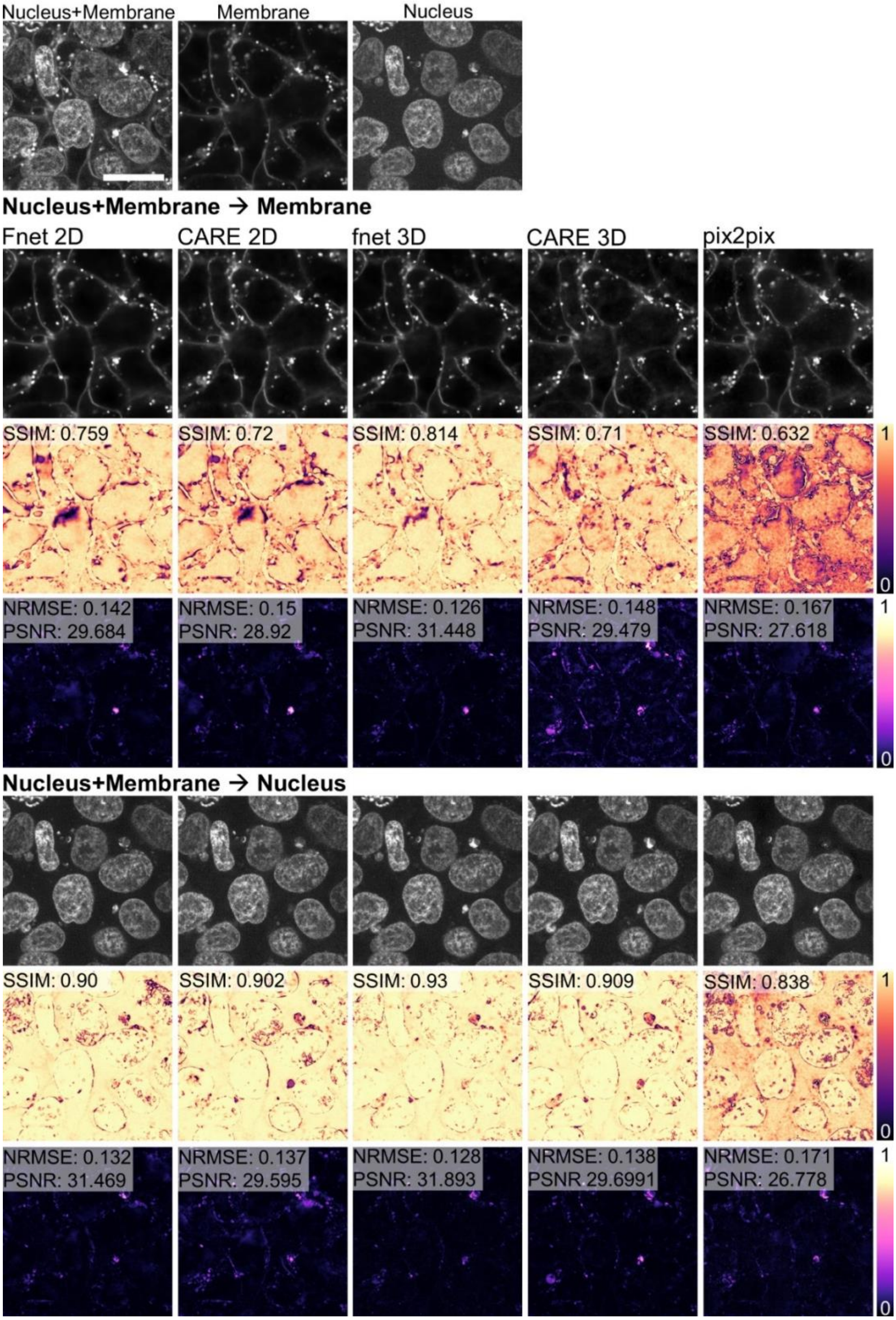

Nucleus+Membrane
Membrane
Nucleus
Nucleus+Membrane → Membrane
Fnet 2D
CARE 2D
fnet 3D
CARE 3D
pix2pix
SSIM: 0.759
SSIM: 0.72
SSIM: 0.814
SSIM: 0.71
SSIM: 0.632
1
0
NRMSE: 0.142
PSNR: 29.684
NRMSE: 0.15
PSNR: 28.92
NRMSE: 0.126
PSNR: 31.448
NRMSE: 0.148
PSNR: 29.479
NRMSE: 0.167
PSNR: 27.618
1
0
Nucleus+Membrane → Nucleus
SSIM: 0.90
SSIM: 0.902
SSIM: 0.93
SSIM: 0.909
SSIM: 0.838
1
0
NRMSE: 0.132
PSNR: 31.469
NRMSE: 0.137
PSNR: 29.595
NRMSE: 0.128
PSNR: 31.893
NRMSE: 0.138
PSNR 29.6991
NRMSE: 0.171
PSNR: 26.778
1
0

***Figure 44 – Unmixing the membrane and nuclear channels from a two-channel composite** – Top: The images represent the same example image pair as the above panels. The input image contains the composite channels, with the middle and right images showing the target channels of the respective unmixing tasks. Middle panel: results of the first unmixing task for the composite image, with predictions on top and SSIM and RSE error maps below representing the errors between predictions and target of the same label (membrane). Lower panel: To be read as the middle panel, with predictions and error maps for the second task (Nucleus) (scale: 20µm).*

IV. 2. 2. e. Unmixing membrane + mitochondria

The omission of the nuclear label from the composite has a similar effect to the unmixing performance as the previous example. The recovery of the membrane label visually appears improved against the prediction from the full composite image, with higher similarity and lower error of predictions to ground-truth for all models (Fig. 45 – top panel). The performance of the models for the membrane recovery in this task is better than in the previous task, with the models performing better when the nuclear label is not present in the dataset. The CARE models show this quite clearly, with the scores approaching those of the fnet models more than in the previous experiment (compare Fig. 44 and 45).

The mitochondrial structure is significantly better restored in this challenge than in the three-channel task for all models. The predictions now include many of the smaller details which were missed when unmixing from the full composite image (Fig. 45 – lower panel). The error maps and metrics show that even the 2D methods now clearly outperform the 3D methods from the full experiment (Fig. 43), with improvements in both background and structural unmixing. While some larger artefacts are still present within the mitochondrial network in the predictions from the 2D methods, these are much smaller in the 3D methods, which likely contributes to their improvement. However, the artefacts remain mostly inside the mitochondrial structures, unlike the membrane predictions where artefacts appear to surround the predicted structure while the membrane itself is predicted with high similarity.

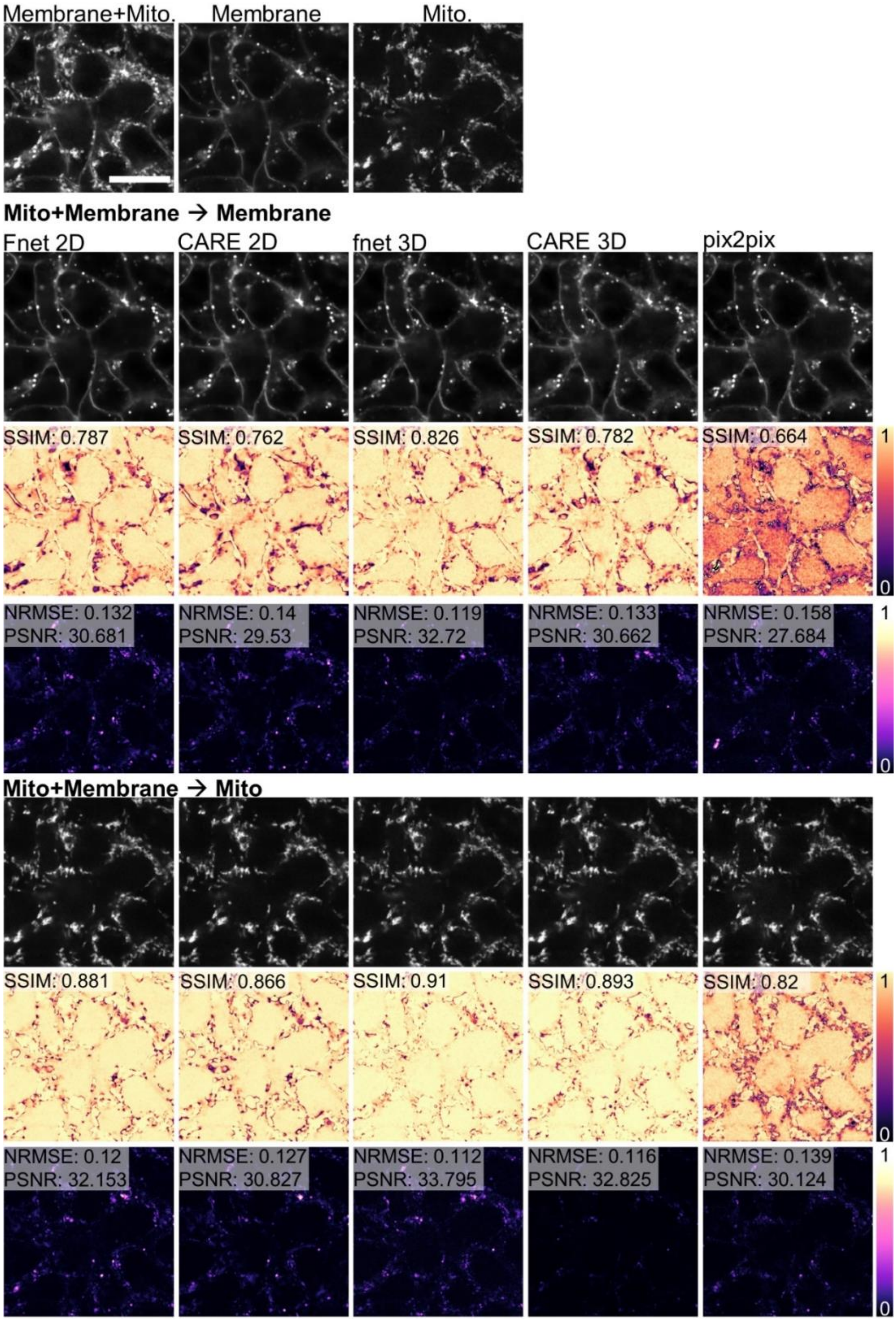

Membrane+Mito.
Membrane
Mito.
Mito+Membrane → Membrane
Fnet 2D
CARE 2D
fnet 3D
CARE 3D
pix2pix
SSIM: 0.787
SSIM: 0.762
SSIM: 0.826
SSIM: 0.782
SSIM: 0.664
1
0
NRMSE: 0.132
PSNR: 30.681
NRMSE: 0.14
PSNR: 29.53
NRMSE: 0.119
PSNR: 32.72
NRMSE: 0.133
PSNR: 30.662
NRMSE: 0.158
PSNR: 27.684
1
0
Mito+Membrane → Mito
SSIM: 0.881
SSIM: 0.866
SSIM: 0.91
SSIM: 0.893
SSIM: 0.82
1
0
NRMSE: 0.12
PSNR: 32.153
NRMSE: 0.127
PSNR: 30.827
NRMSE: 0.112
PSNR: 33.795
NRMSE: 0.116
PSNR: 32.825
NRMSE: 0.139
PSNR: 30.124
1
0

Figure 45 – Unmixing the membrane and mitochondrial channels of a two-channel composite, *to be read as Fig. 44. Middle panel: results of the first unmixing task for the composite image, with predictions on top and SSIM and RSE error maps below representing the errors between predictions and target of the same label (Membrane). Lower panel: To be read as the middle panel, with predictions and error maps for the second task (Mitochondria) (scale: 20µm).*

IV. 2. 2. f. Unmixing nucleus + mitochondria

Unmixing of nuclei from a composite of mitochondria and nuclei appears slightly less accurate visually than from the membrane-nucleus image (Compare Fig. 46 and 44). Here, many of the smaller areas of nuclear label are not recovered, even by the fnet 3D method. The error maps and metrics conversely suggest a slight overall improvement in unmixing nuclei from mitochondria than from the membranes, except for fnet 3D which surprisingly deteriorates for this task, at least for the example image. The error maps, especially the RSE maps show that the models struggle to effectively remove the mitochondrial label from the 'background' to recover the nuclei (Fig. 46 – top panel: bottom row). This is particularly pronounced in pix2pix which deteriorates dramatically compared to the same task for the membrane-nucleus image. The nuclei themselves in all model predictions (especially on the right side of the example image) have many details of their interior misrepresented, which is not obvious from the predictions but clear on the SSIM maps (Fig. 46 – top panel: middle row).

The mitochondrial channel is visually relatively well recovered from the composite, but in contrast to the mitochondrial and membrane composite the predictions of most models again miss the smaller mitochondrial structures (Compare Fig. 46 and 45). However, in contrast to the former experiment, the main contribution to the error appears to be the poor removal of the nuclear signal from the background, notable in the SSIM maps where the area occupied by the nuclei has a comparatively low similarity to the ground-truth compared with the result from unmixing mitochondria from the membrane (Compare Fig. 46 and 45). The mitochondrial structures themselves seem to have a higher similarity in the predictions and with fewer errors than for the previous task. In contrast to the reverse unmixing task (Fig. 46 - top panel), in this task (Fig. 46 – lower panel) the 3D models perform clearly better than the 2D models, and fnet 3D better than CARE 3D as in the previous examples.

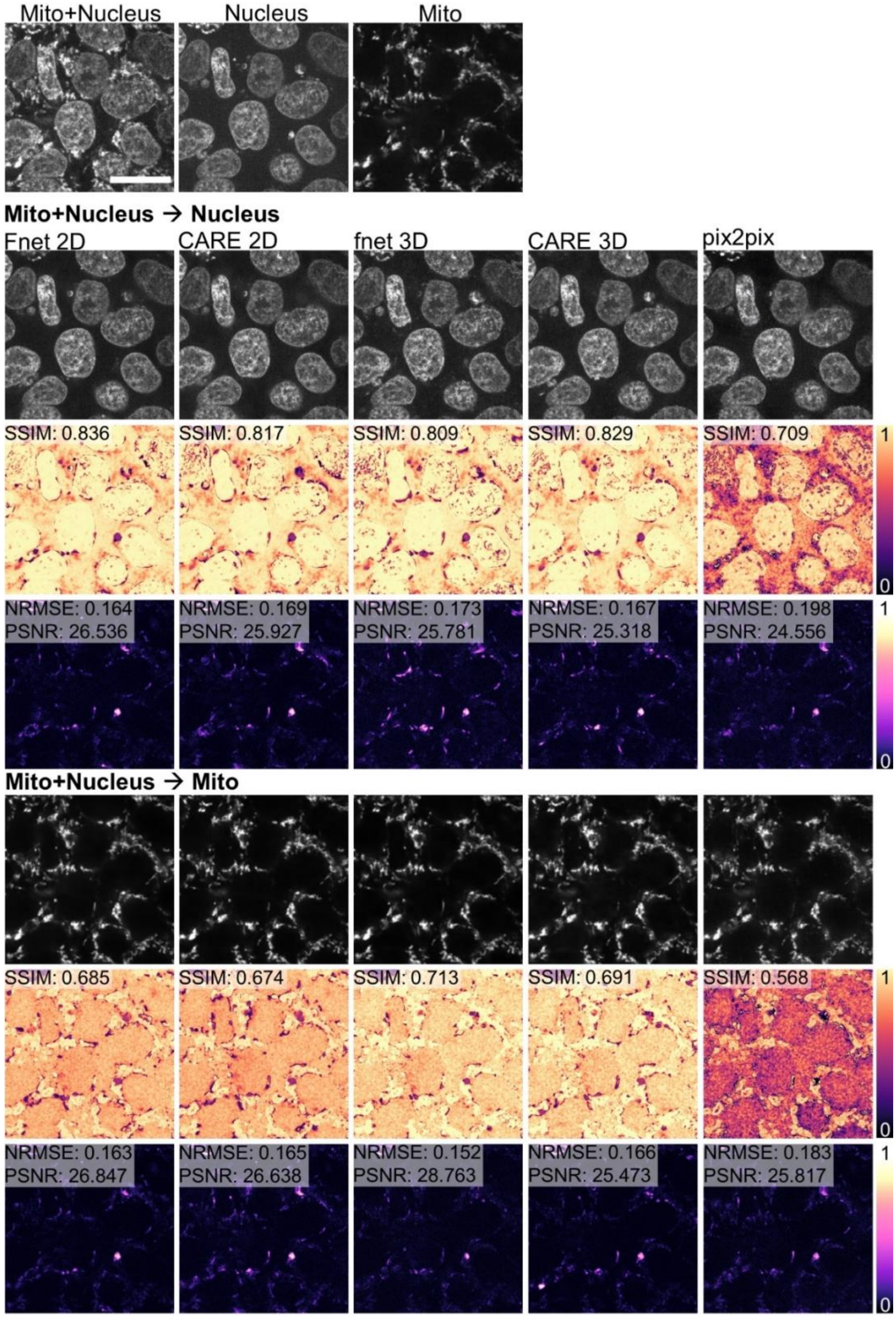

Mito+Nucleus
Nucleus
Mito
Mito+Nucleus → Nucleus
Fnet 2D
CARE 2D
fnet 3D
CARE 3D
pix2pix
SSIM: 0.836
SSIM: 0.817
SSIM: 0.809
SSIM: 0.829
SSIM: 0.709
1
0
NRMSE: 0.164
PSNR: 26.536
NRMSE: 0.169
PSNR: 25.927
NRMSE: 0.173
PSNR: 25.781
NRMSE: 0.167
PSNR: 25.318
NRMSE: 0.198
PSNR: 24.556
1
0
Mito+Nucleus → Mito
SSIM: 0.685
SSIM: 0.674
SSIM: 0.713
SSIM: 0.691
SSIM: 0.568
1
0
NRMSE: 0.163
PSNR: 26.847
NRMSE: 0.165
PSNR: 26.638
NRMSE: 0.152
PSNR: 28.763
NRMSE: 0.166
PSNR: 25.473
NRMSE: 0.183
PSNR: 25.817
1
0

Figure 46 – Unmixing the nuclear and mitochondrial channels from a two-channel composite *– to be read as Fig. 44. Middle panel: results of the first unmixing task for the composite image, with predictions on top and SSIM and RSE error maps below representing the errors between predictions and target of the same label (Nuclei). Lower panel: To be read as the middle panel, with predictions and error maps for the second task (Mitochondria) (scale: 20μm).*

IV. 2. 2. g. Quantitative analysis

Since the number of test images was relatively high and only individual image slices can be analysed visually at one time, the quantitative assessment provided by the average scores of the error metrics over the entire images is crucial in this experiment to verify if the observations made on individual slices hold for the entire test dataset which consisted of 7 stacks with 32 slices each, making the full test dataset contain 224 images.

The analysis was performed for each label, with models compared for each condition which resulted in the prediction of the target label. This allows a comparison of the same method on different training datasets and a comparison of different methods trained on the same training datasets. The analysis also allows insights into which labels were easier to unmix and which ones were more difficult and whether there are differences in performance for certain tasks by certain models. The results of this analysis are displayed in Fig. 47A.

The first key observation from the quantitative analysis of the error metrics is that they appear to agree with each other. This can be concluded from the performance of the different methods and the performance of the models on different tasks ranked in the same order by all metrics, with few exceptions. This suggests that the interpretation of model performance is invariant to the error metrics used for evaluation. Hence, in some of the analysis below, the main metric used for reporting will be SSIM as a proxy for the quality of the models. However, there are some differences in how well the metrics detect differences between the models. Specifically, the NRMSE scores appear to be less sensitive to differences between models in the nuclear recovery task (Fig. 47A – middle row, left). For example, the differences in the NRMSE distributions for the Membrane-Nucleus-to-nucleus and the Mito-Nucleus-to-nucleus experiments in CARE 3D are indistinguishable with almost identical means, whereas the SSIM and PSNR distributions for the same datasets are clearly separable.

Next, it can be observed that as expected, the models generally perform better, i.e. with higher SSIM, lower NRMSE and higher PSNR, when fewer labels need to be removed from the input image. There appears to be one exception to this rule. The fnet 3D model trained for the recovery of nuclear signals which appear to be better at recovering the nuclear label when membrane is present in the input image than when it is not. Although an outlier, this difference is not significant (p=0.37, Mann-Whitney U-Test).

The next key observation is that the nuclear label appears to be the one that is best recovered when comparing only models trained to predict one label from a compound image of three channels, with average SSIM scores in the range of 0.66, 0.67, 0.68 and 0.74 for CARE 2D, 3D and fnet 2D, 3D, respectively, and 0.54 for pix2pix. However, only fnet 3D and pix2pix stand out significantly, at the upper and lower end of performance, respectively, with the other results not significantly different from one another (p<0.01, two-tailed Mann Whitney U-test). The average SSIM scores for the recovery of the membrane channel 0.46, 0.46, 0.5, 0.56 and 0.27 and for the mitochondrial channel 0.39, 0.44, 0.42, 0.5 and 0.3 for CARE 2D, CARE 3D, fnet 2D, fnet 3D and pix2pix, respectively. The significance tests of the differences are shown in Fig. 47B.

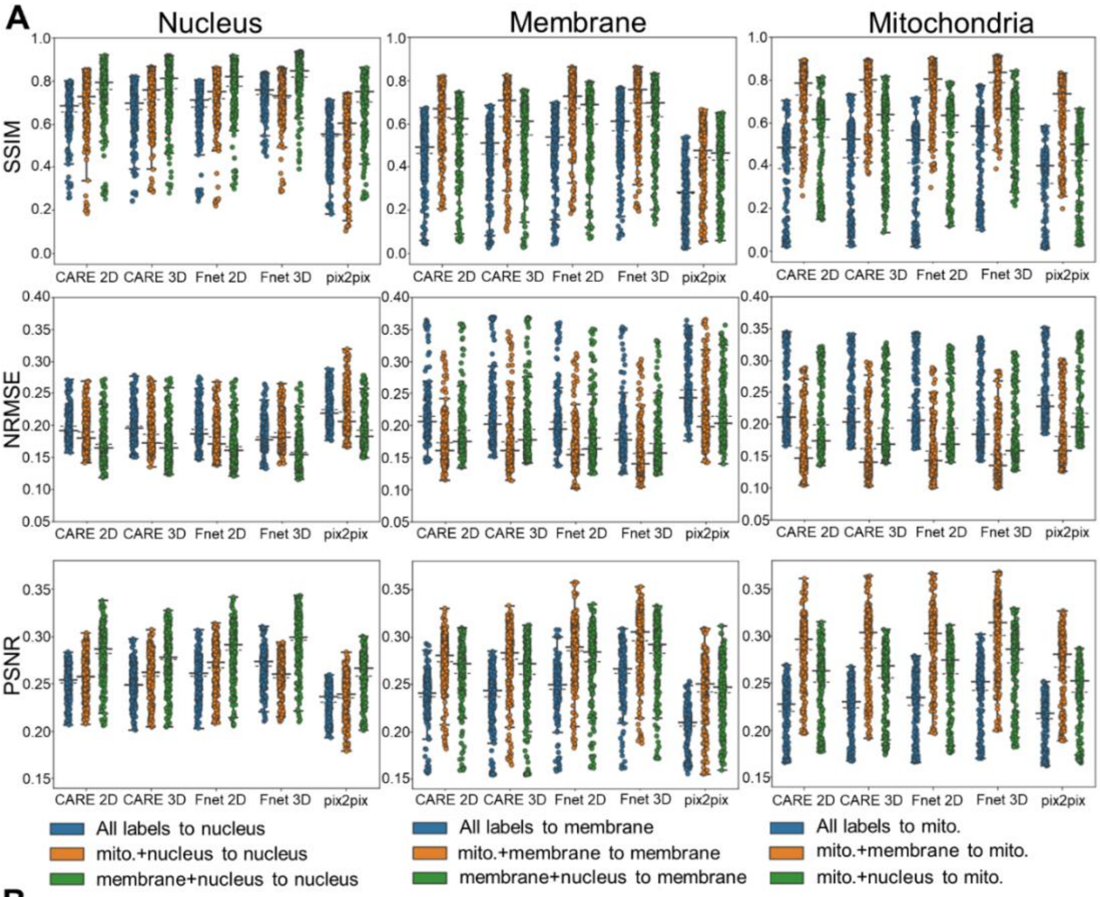
Although the nucleus is the label that is most easily recovered, it appears to be the most difficult to remove from the image. In the two-label to one-label experiments, the presence of the nuclear label made the recovery of another label worse than if the membrane or mitochondria needed to be removed from the image. In turn, this means that membrane and mitochondria are more easily ‘unmixed’ from one another than either is from the nuclear label. The membrane label appears to be the easiest to remove from the images. When the membrane was the structure to be removed from either nuclear or mitochondrial structures, the models showed stronger performance than when removing the other respective labels. The prediction of nucleus from membrane-nucleus composites was the most successful operation of all the tested conditions, with average SSIM on this task for CARE 2D and 3D and fnet 2D and 3D being 0.76, 0.77, 0.78, and 0.82, respectively, and 0.7 for pix2pix (see Fig. 7B for significance of these results).

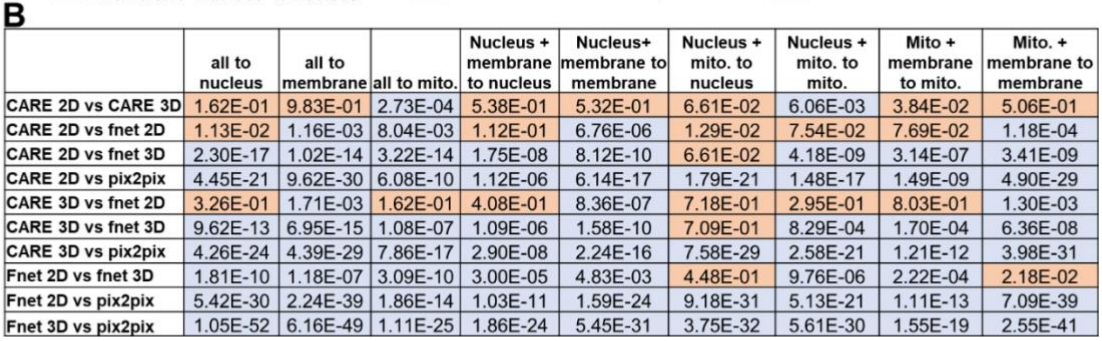
In terms of the performance of the different models against each other, the quantitative analysis supports the observations made when visually inspecting the predictions and error maps of the models. The first key observation is that the U-net based methods perform better than the GAN-based method pix2pix. As mentioned above, it is difficult to interpret this result as the pix2pix method was trained on 8-bit

.pngs while the other models were trained 16-bit .tifs, giving significantly less information per image to train on for this model. Hence, this result is somewhat expected which is why it is also treated apart from the other models.

The other observation is that the 3D models for the other two methods, Fnet and CARE, appear to be generally better at the unmixing task than the 2D models of the same methods. However, for CARE models, these performance differences are not always significant with $p<0.01$ whereas they are for most fnet tasks (Fig. 47B).

Finally, the visual observation that the fnet methods appear to offer a slight improvement over the CARE methods for this task can also be supported by the quantitative analysis performed here. Both fnet methods outrank the CARE methods, with fnet 2D often slightly superior to CARE 3D's predictions. This difference is particularly pronounced for any predictions of membrane labels where fnet 2D consistently and significantly outperforms CARE 3D, whereas for the other tasks, the overall quality of the predictions is nearly indistinguishable between these methods. fnet 3D is better than all other methods under all conditions, except for the exceptional case mentioned above.





	all to nucleus	all to membrane	all to mito.	Nucleus + membrane to nucleus	Nucleus+ membrane to membrane	Nucleus + mito. to nucleus	Nucleus + mito. to mito.	Mito + membrane to mito.	Mito. + membrane to membrane
CARE 2D vs CARE 3D	1.62E-01	9.83E-01	2.73E-04	5.38E-01	5.32E-01	6.61E-02	6.06E-03	3.84E-02	5.06E-01
CARE 2D vs fnet 2D	1.13E-02	1.16E-03	8.04E-03	1.12E-01	6.76E-06	1.29E-02	7.54E-02	7.69E-02	1.18E-04
CARE 2D vs fnet 3D	2.30E-17	1.02E-14	3.22E-14	1.75E-08	8.12E-10	6.61E-02	4.18E-09	3.14E-07	3.41E-09
CARE 2D vs pix2pix	4.45E-21	9.62E-30	6.08E-10	1.12E-06	6.14E-17	1.79E-21	1.48E-17	1.49E-09	4.90E-29
CARE 3D vs fnet 2D	3.26E-01	1.71E-03	1.62E-01	4.08E-01	8.36E-07	7.18E-01	2.95E-01	8.03E-01	1.30E-03
CARE 3D vs fnet 3D	9.62E-13	6.95E-15	1.08E-07	1.09E-06	1.58E-10	7.09E-01	8.29E-04	1.70E-04	6.36E-08
CARE 3D vs pix2pix	4.26E-24	4.39E-29	7.86E-17	2.90E-08	2.24E-16	7.58E-29	2.58E-21	1.21E-12	3.98E-31
Fnet 2D vs fnet 3D	1.81E-10	1.18E-07	3.09E-10	3.00E-05	4.83E-03	4.48E-01	9.76E-06	2.22E-04	2.18E-02
Fnet 2D vs pix2pix	5.42E-30	2.24E-39	1.86E-14	1.03E-11	1.59E-24	9.18E-31	5.13E-21	1.11E-13	7.09E-39
Fnet 3D vs pix2pix	1.05E-52	6.16E-49	1.11E-25	1.86E-24	5.45E-31	3.75E-32	5.61E-30	1.55E-19	2.55E-41

***Fig. 47 – Results from the QC step for all unmixing tasks on the full datasets and significance evaluation** – **A)** Each row of boxplots shows the evaluation of one error metric, each column represents the target label of the respective unmixing experiments (see legends below). The 3D models performed predictions on the full stacks, which were then split into slices for analysis. The whiskers encompass the range between the 1st and 3rd quartiles of the distributions, with the median shown as solid and the mean as a dotted black line. **B)** p-values calculated via Mann-Whitney U-test, evaluated by method and task. P-values < 0.01 and p-values > 0.01 shaded in blue and red, respectively.*

Another notable feature of the quantitative analysis of the full test dataset is that there is a considerable spread in the values of the quality metrics. Although the median and average performance of the models shows relatively high similarities of the predictions to the targets for some unmixing tasks, even in the best performing models there appear to be several images which the model performs poorly for. Even in one of the best fnet models, trained to detect nuclei in the composite of membrane and nuclei channels, several images only have a similarity of 0.2 to 0.4, suggesting relatively poor performance. The NRMSE and PSNR metrics show similarly low performance for some images. For the task of unmixing to be considered successfully completed, it is important to understand why the models perform poorly on some images and conversely when they perform well. A likely explanation in this case is that the models perform better on images that lie in the central plane of the cell, with poorer performance in regions that lie outside this region and are comparably blurry. However, similarly it must be ensured that the opposite is not true, i.e. models do not perform better on relatively trivial tasks, i.e. predicting a blurry image from another blurry image with high similarity, in which case the unmixing task may not be considered successfully completed.

The data gathered in the quality control section of the notebooks can be used to explore this question, as the metrics are evaluated and stored for each individual image. To determine which images the models perform better and more poorly on, the .csv files exported in the QC step for all models were searched for the images with the lowest SSIM, highest NRMSE and lowest PSNR values to determine the images yielding the worst performance from the models and the opposite (highest SSIM, lowest NRMSE, highest PSNR) to determine which yield the best. Since the training and quality control was performed for different datatypes for the U-net based models (CARE 2D/3D and fnet 2D/3D), using 16-bit .tif images, and the pix2pix model, using 8-bit .pngs, the results of the analysis are presented separately in Fig. 48, for clarity.

The results of the search for the best and worst performance on the datasets for the CARE and fnet models are shown in Fig. 48A. The high proportion of slices from the tops and bottoms of the stacks yielding the lowest performances suggests that as hypothesized, the models perform poorer on areas of the stacks that are out of focus. However, this may be contradicted by the presence of some ‘extreme’ slices in the columns collecting the best models' performances, although notably never from the same stack. In Fig. 48B, two examples are shown for images yielding the lowest and highest performance for a task (membrane-nucleus to nucleus). These slices were chosen because the models agree broadly that these images yield extremely high and low performances across metrics as they both represent the outermost slices of their image stack. The image yielding the worst performances is blurry and provides little detail in the input or the target for the models to unmix. The image with the best performance shows all the main structures expected in the image, in input and target and yields visually accurate predictions (error maps are not shown in this example). The fact that despite the differing content, the images both represent the edges of their stacks suggests that the stacks were not completely aligned in the z-direction during acquisition. When creating the dataset for these experiments, this may have led to some stacks being cropped nearer to the central plane of the cells than others. Hence, it can be assumed that the upper and lower slices yielding high SSIM values in the QC are similar to the one shown in the figure, and similar to the more 'central' slices which make up the majority of the images resulting in the peak-performances for the models.

A similar behaviour can be observed for the pix2pix models, although these do not agree with the other models in terms of the identities of the images yielding the poorest and best performances. Here, the example images are chosen for the agreement across all metrics that these images represent the extremes for this task. As before, the model performs worse on the blurry image and better on the image containing clear structures. Unlike the above models the pix2pix model for the membrane-nucleus to membrane task predicts significant and obvious artefacts in the blurry image which significantly differ from the target which may explain why this prediction is ranked relatively low.

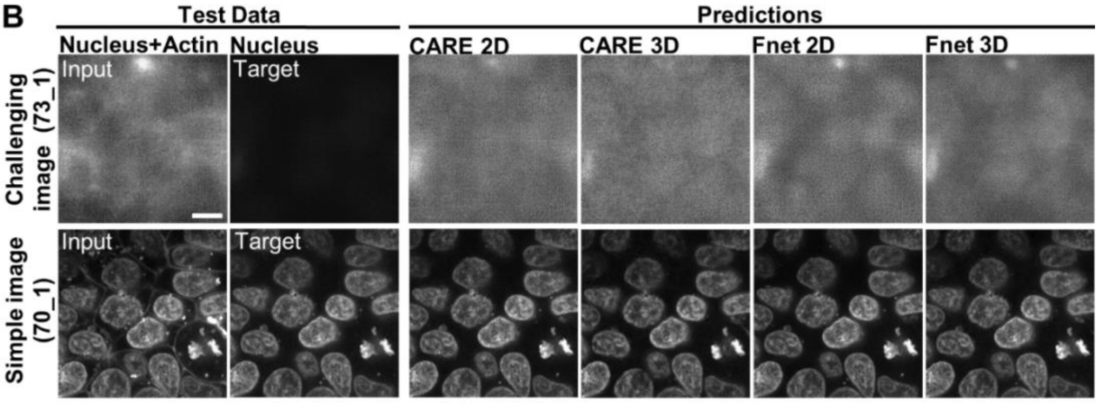
Although only one example image was chosen for the CARE/fnet and pix2pix models, respectively, the observation that the 'extreme' slices lead to poor performance according to the quality metrics suggests that the models generally perform better on the images which provide richer detail. This means that for the key task of unmixing channels from images with detailed structural information, the models may perform slightly better than the averages or medians of the above-shown distributions suggest.

Another interesting observation is that the metrics often agree on the images that yield the best and worst performances from the models for specific tasks. This somewhat validates the choice of interpreting the above performance primarily using only one metric, the SSIM metric, although the metrics do not measure the same quantities.

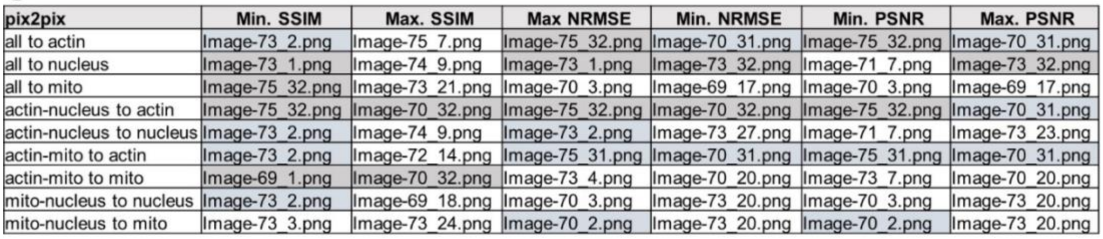
A

	Min. SSIM	Max. SSIM	Max. NRMSE	Min. NRMSE	Min. PSNR	Max. PSNR
CARE 2D						
all to actin	Image-75_26.tif	Image-70_31.tif	Image-75_26.tif	Image-70_28.tif	Image-75_26.tif	Image-70_28.tif
all to nucleus	Image-73_9.tif	Image-74_10.tif	Image-73_1.tif	Image-73_28.tif	Image-73_1.tif	Image-73_15.tif
all to mito	Image-71_6.tif	Image-73_27.tif	Image-70_3.tif	Image-73_20.tif	Image-70_3.tif	Image-73_19.tif
actin-nucleus to actin	Image-75_30.tif	Image-71_16.tif	Image-75_27.tif	Image-70_29.tif	Image-75_30.tif	Image-70_29.tif
actin-nucleus to nucleus	Image-73_1.tif	Image-70_31.tif	Image-73_1.tif	Image-70_31.tif	Image-73_1.tif	Image-70_31.tif
actin-mito to actin	Image-75_32.tif	Image-70_28.tif	Image-75_32.tif	Image-70_28.tif	Image-75_32.tif	Image-70_26.tif
actin-mito to mito	Image-69_1.tif	Image-70_31.tif	Image-73_2.tif	Image-70_23.tif	Image-73_2.tif	Image-70_19.tif
mito-nucleus to nucleus	Image-73_6.tif	Image-69_14.tif	Image-73_6.tif	Image-73_21.tif	Image-73_6.tif	Image-73_17.tif
mito-nucleus to mito	Image-70_5.tif	Image-73_27.tif	Image-70_3.tif	Image-73_19.tif	Image-70_3.tif	Image-73_18.tif
CARE 3D						
all to actin	Image-75_32.tif	Image-71_16.tif	Image-75_26.tif	Image-70_27.tif	Image-75_26.tif	Image-73_10.tif
all to nucleus	Image-73_2.tif	Image-74_9.tif	Image-73_1.tif	Image-73_31.tif	Image-73_1.tif	Image-73_21.tif
all to mito	Image-75_32.tif	Image-73_22.tif	Image-75_31.tif	Image-73_21.tif	Image-75_31.tif	Image-69_17.tif
actin-nucleus to actin	Image-75_32.tif	Image-71_12.tif	Image-75_26.tif	Image-70_27.tif	Image-75_26.tif	Image-73_15.tif
actin-nucleus to nucleus	Image-73_1.tif	Image-70_24.tif	Image-73_1.tif	Image-70_25.tif	Image-73_1.tif	Image-70_23.tif
actin-mito to actin	Image-75_32.tif	Image-71_18.tif	Image-75_32.tif	Image-70_27.tif	Image-75_32.tif	Image-70_26.tif
actin-mito to mito	Image-69_1.tif	Image-72_15.tif	Image-73_1.tif	Image-70_20.tif	Image-73_1.tif	Image-70_20.tif
mito-nucleus to nucleus	Image-73_4.tif	Image-69_16.tif	Image-73_2.tif	Image-73_32.tif	Image-73_2.tif	Image-73_32.tif
mito-nucleus to mito	Image-75_2.tif	Image-73_27.tif	Image-75_2.tif	Image-73_22.tif	Image-75_2.tif	Image-69_16.tif
fnet 2D						
all to actin	Image-75_32.tif	Image-70_30.tif	Image-75_32.tif	Image-70_29.tif	Image-75_32.tif	Image-70_30.tif
all to nucleus	Image-73_1.tif	Image-72_32.tif	Image-73_1.tif	Image-73_31.tif	Image-73_1.tif	Image-73_18.tif
all to mito	Image-75_32.tif	Image-73_23.tif	Image-70_1.tif	Image-73_18.tif	Image-70_5.tif	Image-70_20.tif
actin-nucleus to actin	Image-75_32.tif	Image-71_16.tif	Image-75_32.tif	Image-70_27.tif	Image-75_32.tif	Image-70_27.tif
actin-nucleus to nucleus	Image-73_1.tif	Image-70_29.tif	Image-73_1.tif	Image-70_32.tif	Image-73_1.tif	Image-70_32.tif
actin-mito to actin	Image-75_32.tif	Image-70_27.tif	Image-75_32.tif	Image-70_27.tif	Image-75_32.tif	Image-70_27.tif
actin-mito to mito	Image-69_1.tif	Image-70_26.tif	Image-73_1.tif	Image-70_21.tif	Image-73_1.tif	Image-70_21.tif
mito-nucleus to nucleus	Image-73_4.tif	Image-69_15.tif	Image-73_1.tif	Image-73_17.tif	Image-73_1.tif	Image-73_17.tif
mito-nucleus to mito	Image-75_31.tif	Image-73_27.tif	Image-75_31.tif	Image-73_17.tif	Image-75_31.tif	Image-73_17.tif
fnet 3D						
all to actin	Image-75_32.tif	Image-71_16.tif	Image-75_32.tif	Image-70_27.tif	Image-75_32.tif	Image-70_27.tif
all to nucleus	Image-73_1.tif	Image-74_9.tif	Image-73_1.tif	Image-73_32.tif	Image-73_1.tif	Image-73_21.tif
all to mito	Image-75_32.tif	Image-73_23.tif	Image-70_1.tif	Image-73_20.tif	Image-70_1.tif	Image-73_17.tif
actin-nucleus to actin	Image-75_32.tif	Image-72_10.tif	Image-75_32.tif	Image-72_10.tif	Image-75_31.tif	Image-70_27.tif
actin-nucleus to nucleus	Image-73_1.tif	Image-70_29.tif	Image-73_1.tif	Image-70_29.tif	Image-73_1.tif	Image-70_25.tif
actin-mito to actin	Image-75_32.tif	Image-70_27.tif	Image-75_32.tif	Image-70_28.tif	Image-75_32.tif	Image-70_28.tif
actin-mito to mito	Image-69_1.tif	Image-72_15.tif	Image-69_1.tif	Image-70_21.tif	Image-69_1.tif	Image-70_20.tif
mito-nucleus to nucleus	Image-73_4.tif	Image-69_15.tif	Image-73_1.tif	Image-73_31.tif	Image-73_1.tif	Image-73_32.tif
mito-nucleus to mito	Image-75_32.tif	Image-73_27.tif	Image-75_32.tif	Image-73_21.tif	Image-75_32.tif	Image-73_17.tif

B



C



pix2pix	Min. SSIM	Max. SSIM	Max NRMSE	Min. NRMSE	Min. PSNR	Max. PSNR
all to actin	Image-73_2.png	Image-75_7.png	Image-75_32.png	Image-70_31.png	Image-75_32.png	Image-70_31.png
all to nucleus	Image-73_1.png	Image-74_9.png	Image-73_1.png	Image-73_32.png	Image-71_7.png	Image-73_32.png
all to mito	Image-75_32.png	Image-73_21.png	Image-70_3.png	Image-69_17.png	Image-70_3.png	Image-69_17.png
actin-nucleus to actin	Image-75_32.png	Image-70_32.png	Image-75_32.png	Image-70_32.png	Image-75_32.png	Image-70_31.png
actin-nucleus to nucleus	Image-73_2.png	Image-74_9.png	Image-73_2.png	Image-73_27.png	Image-71_7.png	Image-73_23.png
actin-mito to actin	Image-73_2.png	Image-72_14.png	Image-75_31.png	Image-70_31.png	Image-75_31.png	Image-70_31.png
actin-mito to mito	Image-69_1.png	Image-70_32.png	Image-73_4.png	Image-70_20.png	Image-73_7.png	Image-70_20.png
mito-nucleus to nucleus	Image-73_2.png	Image-69_18.png	Image-70_3.png	Image-73_20.png	Image-70_3.png	Image-73_20.png
mito-nucleus to mito	Image-73_3.png	Image-73_24.png	Image-70_2.png	Image-73_20.png	Image-70_2.png	Image-73_20.png

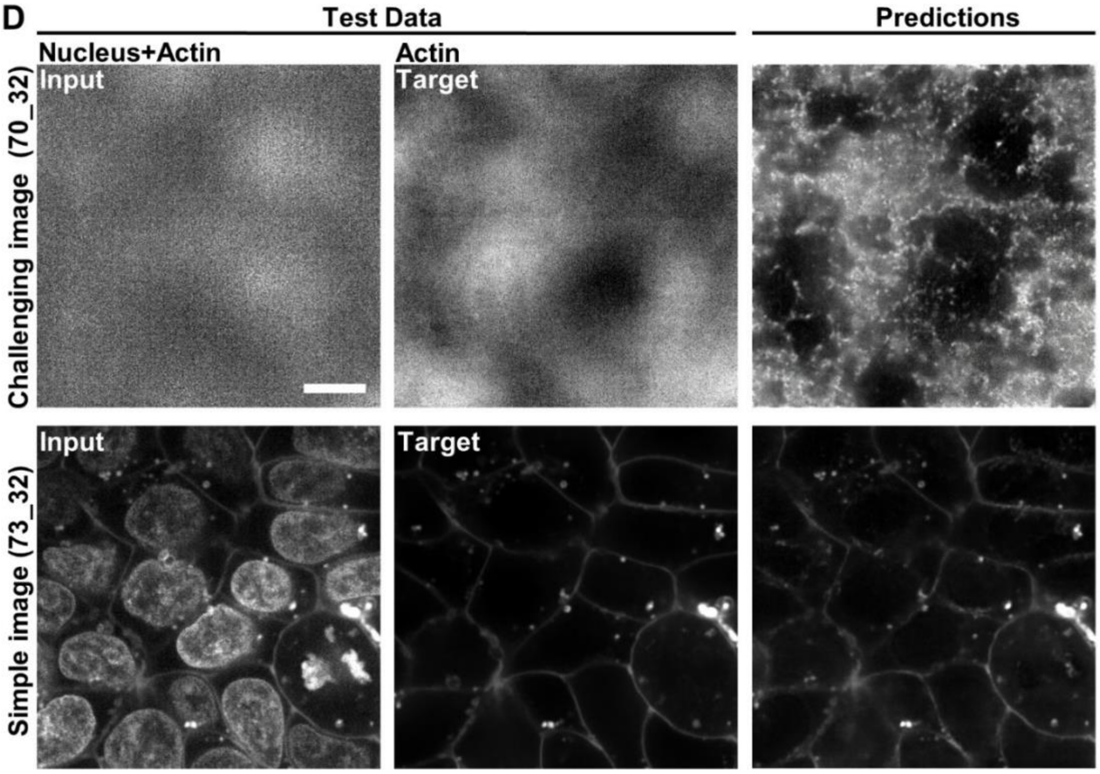


Figure 48 – Worst and best performance examples of the unmixing tasks – A) *The table shows the names of the images which yielded the worst and best performances of the models by task and metric. The image names can be interpreted as: Image-StackID_slice.tif. The highlighted areas indicate the images which are at the highest or lowest slice in their stack (in grey) or the second-highest or second lowest (blue).* ***B)*** *Top Row: Image identified from above table representing a case of low performance of the models; Lower Row: Image identified from above table representing a case of high performance of the models.* ***C)*** *Table to be interpreted as above for performance of the pix2pix model.* ***D)*** *As C for the pix2pix model (scales: 10µm)*

IV. 3. Finetuning

The above results suggest that the DL models implemented in the ZeroCostDL4Mic notebooks can perform the unmixing task, with differences depending on the combination of labels in the composite and the target image. To be of interest as a tool for a user, e.g. in freeing up an imaging channel in their imaging experiments, it is important to maximise the performance of the chosen tool, to ensure that the predictions are as rich in information and contain as few errors as possible. In the initial experiment, the models were trained with as much of the same parameters as possible to allow an initial comparison between the different methods while using settings that would allow the notebooks to be used within the limitations of Colab. However, the chosen parameters which were partly informed by the results in chapter 2 could have been ill-suited for the optimal training performance of the methods, and thus, further adjustments could potentially improve the performance further and potentially remove some of the differences between the different methods.

However, fine-tuning all the above models is challenging for two reasons. As explained previously, exhaustive fine-tuning of the parameters available in the notebooks is beyond the scope of the project, as the number of possible combinations for the parameters is very high and thus difficult to realise even for a single method. For fine-tuning, I focused on key parameters which were expected to improve the performance for each model instead of tuning every parameter in the notebooks, similar to and inspired by the previous chapter. A further difficulty encountered in this experiment is that due to the size of the test dataset of 224 images, the QC step is memory intensive, as it leads to 224 predictions and at least 2 error maps for each prediction resulting in 672 files for each quality control step for each model. For the pix2pix models, this is increased by an additional factor the size of which depends on the number of epochs the model is trained for. For the pix2pix models trained for this chapter each pix2pix model has 15 checkpoints for each of which QC is performed with the above dataset. This is extremely memory intensive and limited the potential for in-depth QC without needing to remove previously trained models from the Google Drive.

Hence, the focus for finetuning was on either boosting the performance of the best models to investigate the limits of the tools for this task or improving the models for those label combinations for which performance was low. With these considerations, the fine-tuning was performed for the task of Membrane-Nucleus to nucleus, the task which yielded some of the peak scores in the above experiment, and the Membrane-Nucleus to membrane task, which yielded some of the lowest scores.

IV. 3. 1. Finetuning CARE

When comparing the CARE models with one another in the QC section, a notable difference between the CARE 2D and 3D methods was that the former quickly overfit during training while the latter did not (Fig. 49A). Hence, the first measure in finetuning the CARE 2D model was to augment the dataset to give the model more data to train on. Considering that the CARE method creates potentially overlapping patches (see Chapter 2), another reason for the poor performance could be overlaps between patches that lead to overfitting to the training data. Hence, for finetuning the dataset was created by reducing the number of patches per image to 8 and augmenting the dataset by 4, using only rotations by 90 degrees, leading to an effective augmentation by 2. The intention is that this would result in data with fewer overlaps between patches, yet more total data to train with. Reducing the number of patches also has the advantage that it would not break the runtime limitations of the Colab notebook. To test if performance could be improved in this manner, in CARE 3D as well, these augmentations were also performed on the 3D dataset (in the x-y plane) and new models were trained for both methods, with 8 instead of 16 patches per image.

Comparing the loss curves of CARE 2D with and without augmentation suggests that these adjustments do not result in an improvement in either of the tasks (composite to membrane and composite to nucleus) (Fig. 49A, left panel). The predictions also show no clear improvement over the base model (Fig. 49B), suggesting that 2x augmentation does not improve CARE 2D performance for this task. In the 3D model, data augmentation led, first, to a small improvement in the validation loss achieved during training for both labels and, second, to visible improvements in unmixing of the membrane from the composite (Fig. 49B). For the nuclear channel the predictions cannot be distinguished in terms of quality between base model and the model trained on augmented data.

Since it was shown in chapter 2 that CARE performance is better if training is stopped before any overfitting occurs even if this means saving models with a worse validation loss (see Fig. 35), this approach was also tested here. A CARE 2D model was thus trained for only 10 epochs (reducing the number of epochs by half) on the original dataset to test if this could improve performance on the test dataset. While the losses indicate no or minimal overfitting in this training, the predictions do not appear better than the base model (Fig. 49B). The same was not done for CARE 3D, as the models' losses did not diverge under any of the conditions. To test whether other measures could improve CARE 3D performance on this task, the stack size was halved, so that more training data would be sampled per stack. However, reducing the stack depth by half led to a significant deterioration of the performance (Fig. 49B). The membrane reconstruction shows that the nuclei are not thoroughly removed during the unmixing by this model. The nuclear channel reconstruction is closer to the base model but the artefacts within the nuclei are larger, which is evident particularly in the error maps.

The visual results on the example predictions are also reflected in the quantitative analysis over the entire dataset (Fig. 49C). In fact, none of the CARE 2D models trained for these trials significantly improved the performance of the base model. Similarly, the performance of the 3D model only significantly improves after augmentation for the membrane channel ($p<0.001$), but not for the nuclear channel ($p<0.2$). The reduction of stack depth leads to a significant reduction in quality in the CARE 3D model.

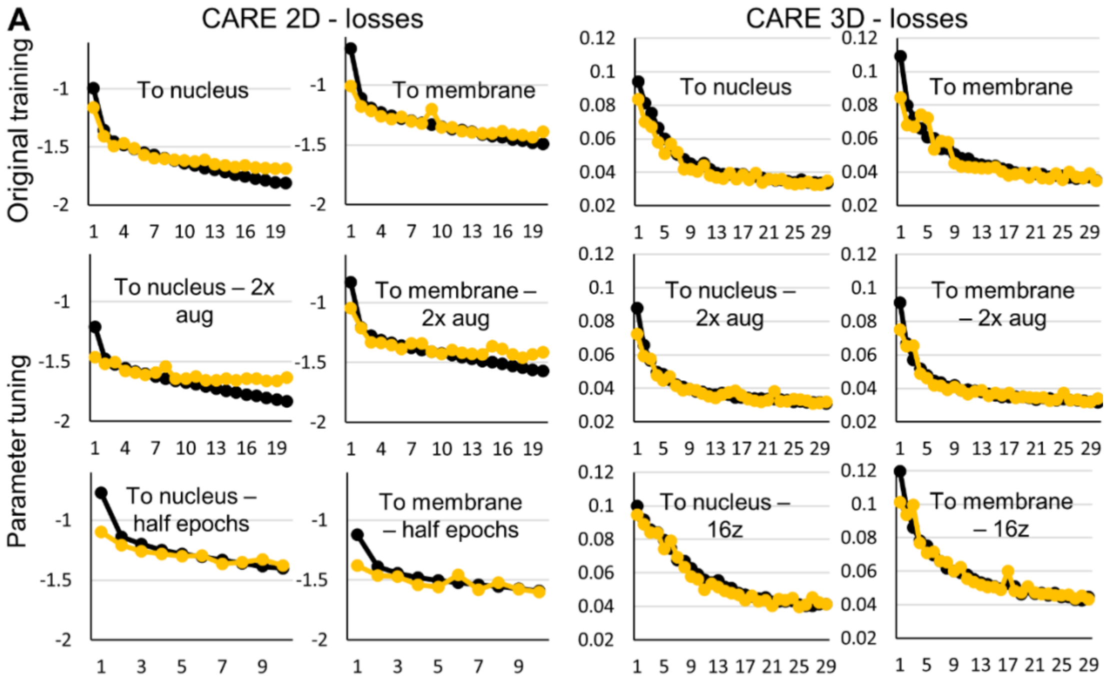


__Figure 49A – Finetuning CARE models for membrane-nucleus unmixing__ – Loss curves for CARE models trained to finetune on the unmixing task, with parameter adjustment shown in the titles (black – training loss, yellow – validation loss).

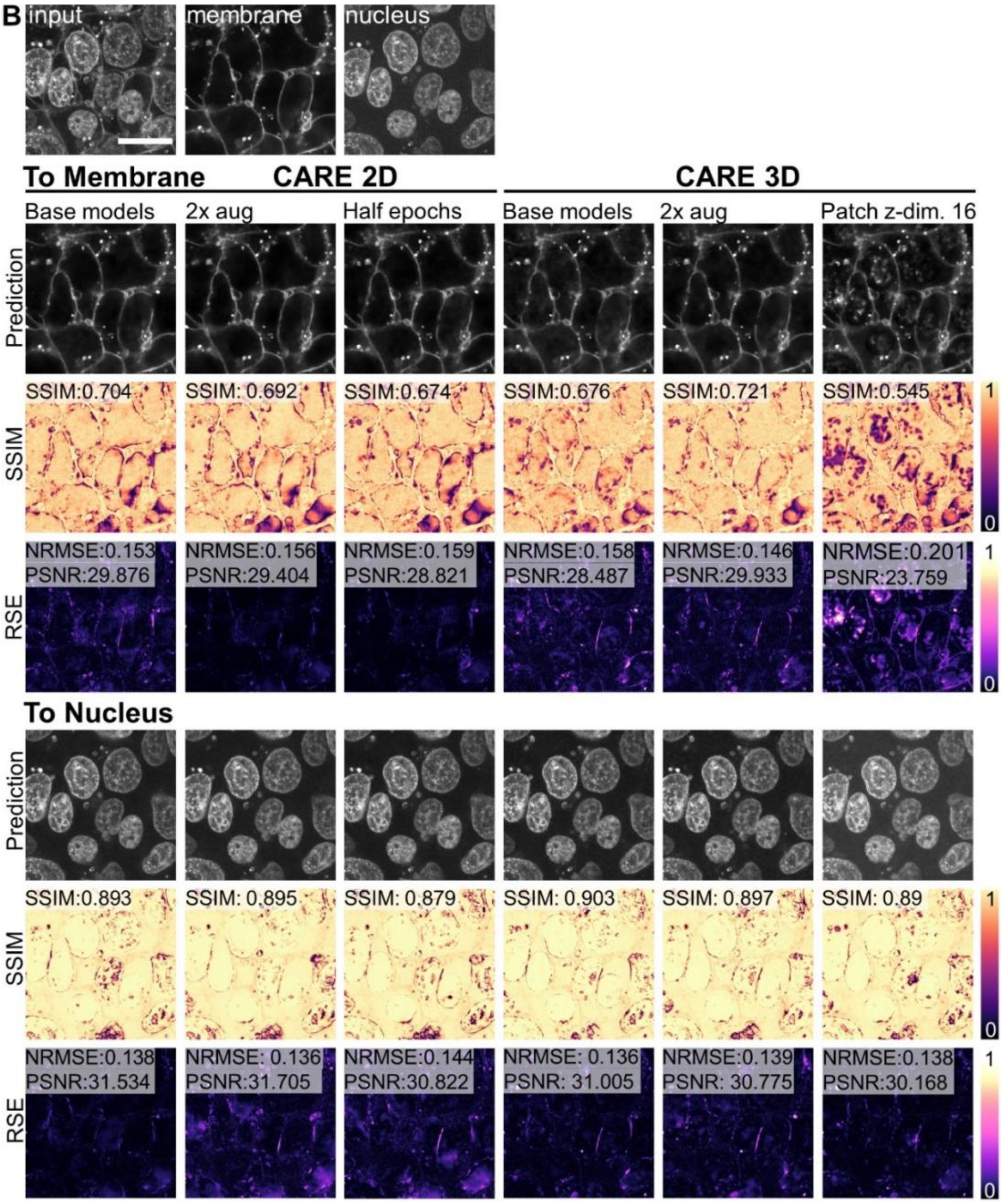

B
input
membrane
nucleus
To Membrane
CARE 2D
CARE 3D
Base models
2x aug
Half epochs
Base models
2x aug
Patch z-dim. 16
Prediction
SSIM
RSE
SSIM:0.704
SSIM: 0.692
SSIM:0.674
SSIM:0.676
SSIM:0.721
SSIM:0.545
1
0
NRMSE:0.153
PSNR:29.876
NRMSE:0.156
PSNR:29.404
NRMSE:0.159
PSNR:28.821
NRMSE:0.158
PSNR:28.487
NRMSE:0.146
PSNR:29.933
NRMSE:0.201
PSNR:23.759
1
0
To Nucleus
Prediction
SSIM
RSE
SSIM:0.893
SSIM: 0.895
SSIM: 0.879
SSIM: 0.903
SSIM: 0.897
SSIM: 0.89
1
0
NRMSE:0.138
PSNR:31.534
NRMSE: 0.136
PSNR:31.705
NRMSE:0.144
PSNR:30.822
NRMSE: 0.136
PSNR: 31.005
NRMSE:0.139
PSNR: 30.775
NRMSE:0.138
PSNR:30.168
1
0

***Figure 49B – Finetuning CARE models for membrane-nucleus unmixing** – Comparison of predictions of CARE before and after changing parameters in the ZeroCostDL4Mic notebooks – Top: composite input (nucleus + membrane) and targets (nucleus, membrane); Middle panel: predictions of membrane channel by CARE models (2D left, 3D right): CARE 2D predictions from: original model (same model as in Fig. 41) (left), a model trained on a twice augmented dataset (middle) and a model trained on fewer epochs (right) than the original model. CARE 3D predictions from: original model (same as Fig. 1), a model trained on a twice augmented dataset and a model trained with training stack depth of 16 slices; Lower panel: predictions of the nucleus channel by CARE models, to be read as middle panel. (scale: 20µm)*

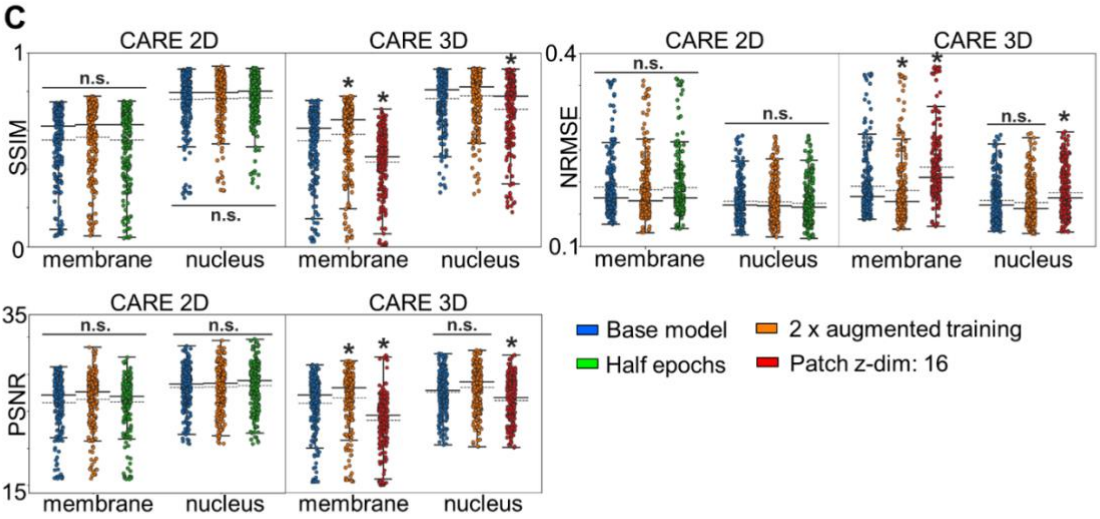


***Figure 49C – Finetuning CARE models for membrane-nucleus unmixing** - QC metrics calculated for the full test dataset, for above models. (The whiskers show limits of the 1st and 3rd quartile of the distributions, median shown as solid and the mean as a dotted black line.) The asterisks indicate where the distributions differ significantly (with p<0.01, according to Mann Whitney U-test) from the base model. (n.s. – not significant).*

IV. 3. 2. Finetuning fnet

In the previous chapter, it was found that the performance of the fnet models for a label-prediction task could be improved by adding steps to the training of the models. To do this, the previously trained models were loaded into the notebook and trained for additional epochs on the same dataset, in both cases doubling the number of steps, to a total number of 97600 for fnet 2D and 24400 for fnet 3D. The fnet notebooks are specifically designed to facilitate retraining on the same dataset because training sessions can easily exceed the runtime of the notebooks, thus enabling the completion even of very long training.

Differences between models are easily detected in the loss curves (Fig. 50A). All models continue to improve their training losses to the dataset, although overfitting again appears significant. However, as noted in the previous chapter, the overfitting seen in the losses may be a peculiarity of fnet and is not a solid predictor of the performance of the models on unseen data. However, when comparing the predictions of the trained models visually, the differences between the initially trained and re-trained models for an example image are very small (Fig. 50B). The differences between all models are almost imperceptible for the membrane channel whereas the nuclear channel appears to show minor improvements for the fnet 3D model which are most visible in the SSIM error maps which show the reduction of some of the darker (low similarity) areas in the prediction of the models trained for additional steps, e.g. in the lower part of the example image.

Although there are small improvements in the average and median scores for the 3D models trained for additional steps, none of the improvements are significant, i.e. with $p<0.01$, although the median SSIM of the predictions vs. ground-truths of the fnet 3D model for the membrane channel differ from the base model with $p<0.05$ (Mann-Whitney U-test), suggesting potentially small improvements for this task. This suggests that the model performs slightly better for the membrane channel than the visual inspection suggested which may simply mean that the example is not entirely representative of the dataset. Although the improvements for the membrane unmixing in fnet 3D may not be significant, inspecting the distributions appears to show that the main small improvements which do occur do so in images for which the base model already performed well, since the lowest values do not improve in the model trained for additional steps. This suggests that the main benefit of additional training indeed lies in improving on those images which could be of most relevance for a potential user (see Fig. 48).

In the 2D models, the distributions of error metrics do not differ significantly, suggesting no improvement by additional training.

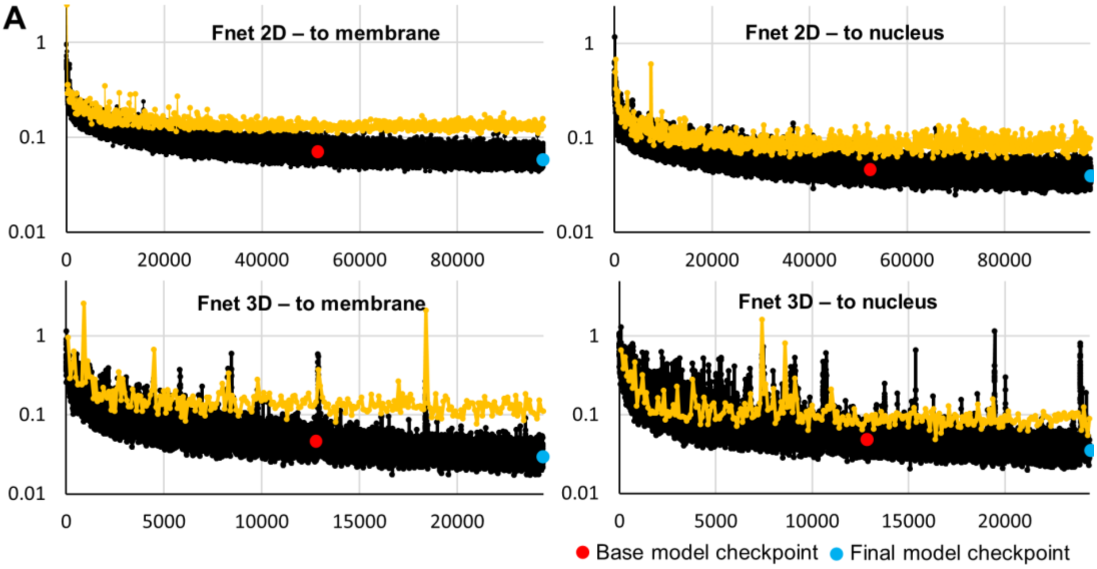


***Figure 50A - Finetuning fnet models for membrane-nucleus unmixing by additional training** – Loss curves of fnet models trained for 2-label unmixing, with checkpoints used for evaluation shown in the curves.*

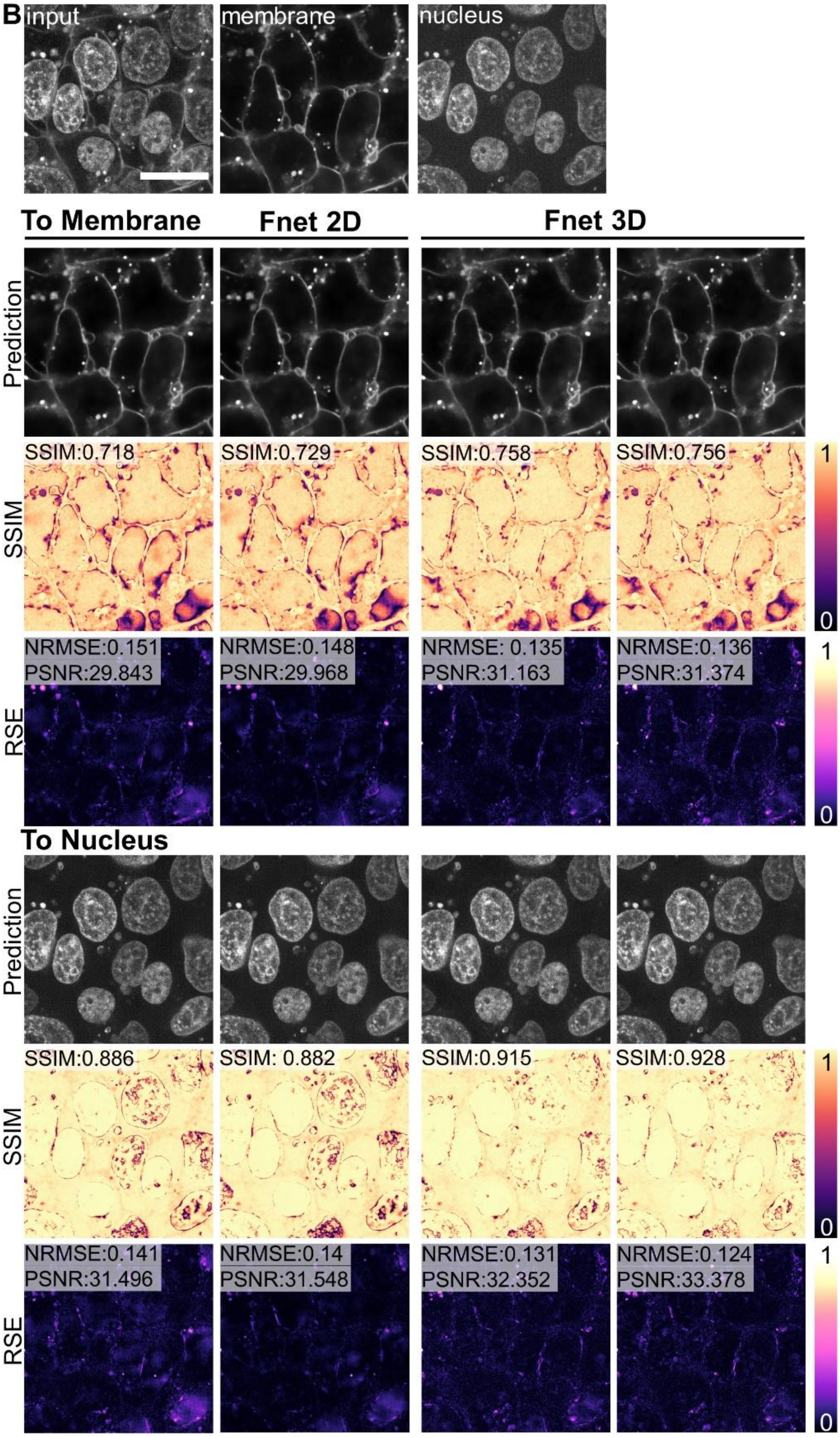

B
input
membrane
nucleus
To Membrane
Fnet 2D
Fnet 3D
Prediction
SSIM
SSIM:0.718
SSIM:0.729
SSIM:0.758
SSIM:0.756
1
0
RSE
NRMSE:0.151
PSNR:29.843
NRMSE:0.148
PSNR:29.968
NRMSE: 0.135
PSNR:31.163
NRMSE:0.136
PSNR:31.374
1
0
To Nucleus
Prediction
SSIM
SSIM:0.886
SSIM: 0.882
SSIM:0.915
SSIM:0.928
1
0
RSE
NRMSE:0.141
PSNR:31.496
NRMSE:0.14
PSNR:31.548
NRMSE:0.131
PSNR:32.352
NRMSE:0.124
PSNR:33.378
1
0

Figure 50B - Finetuning fnet models for membrane-nucleus unmixing by additional training *- Comparison of predictions of fnet before and after training for additional steps in the ZeroCostDL4Mic notebooks – Top: composite input (nucleus + membrane) and targets (nucleus, membrane); Middle panel: Predictions of membrane channel of fnet 2D models (left) and fnet 3D models (right), with original model predictions on the left and model predictions after additional training on the right of the respective models. Lower panel:*

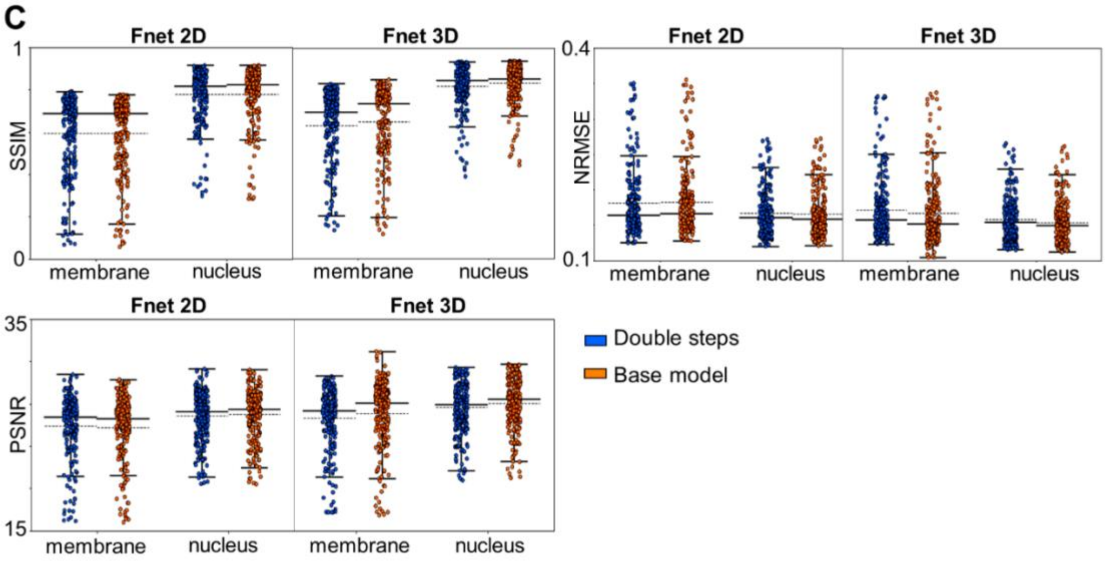


Figure 50C - Finetuning fnet models for membrane-nucleus unmixing by additional training *- QC metrics for the full test dataset for above models. (The whiskers encompass the range between the 1st and 3rd quartile of the distributions, with the median shown as solid and the mean as a dotted black line). The distributions of QC metrics are not significantly different according to the Mann Whitney U-test (with p<0.01), but for the 3D model p<0.05 for the membrane model, and p=0.056 for the nuclear model.*

IV. 3. 3. Finetuning pix2pix

Although the pix2pix model was trained on a dataset of lower bit-depth and a different patch size, this model also has the greatest scope for improvement. However, as mentioned above, the fine-tuning of the pix2pix model is somewhat limited by the memory footprint of the QC for this method in ZeroCostDL4Mic. A notable aspect of the pix2pix method is that it did not accept a patch size smaller than 256x256 for training, suggesting that it requires a relatively large field view compared to CARE and fnet which can work with much smaller patch sizes. Therefore, the main experiment carried out for pix2pix was to increase the patch size by a factor of 2 per dimension, giving a training patch size of 512x512, so that the entire field of view of each image is covered. An additional consideration is which model checkpoint to use for the unmixing task. In pix2pix, unlike the other methods, the learning trajectory is not necessarily one that consistently leads to improvements but can fluctuate greatly. It may also not be suitable to save only model checkpoints with a low discriminator or generator loss as neither may offer the best performance on test data. Instead, the notebook saves a checkpoint every 10 epochs, and in the QC section every model is tested on the test dataset. The average SSIM for each checkpoint is saved and can be plotted to create a pseudo-learning curve. This allows the user to identify the model which scores on average the highest on the test dataset. Although this may not be the ideal solution to find the best model it is justified by the observation made above that one metric may be sufficient to give an insight into a model’s performance, and by the need for a memory-efficient QC step which would make it impossible to save and test a checkpoint for each epoch of training.

Under these considerations, the pix2pix models chosen for this experiment were those with the highest SSIM score over the test datasets which can already give an indication of overall performance. In the task of unmixing membrane from the composite image, the checkpoints with the highest SSIM for the different patch sizes are nearly identical, while for the nucleus the larger patch size appears to lead to a higher peak in the SSIM scores for the best checkpoint (Fig. 51A). Visually, the differences between the models’ predictions are minor (Fig. 51B). The membrane predictions show slightly sharper edges and higher contrast between signal and background, which is confirmed in the error maps which reflect these small improvements. The nuclear channel appears to be worse after training on a larger patch size, with larger and more pronounced artefacts within the nuclei. These impressions are reflected in the overall analysis on the test dataset where the model trained on membranes shows a significant improvement (p<0.01) while there are no significant changes in the models trained to unmix the nucleus (Fig. 51C). Indeed, the medians and averages for the quality metrics in this model are reduced compared to the base model, suggesting an even slightly worse performance in the larger patch size, which is surprising because the analysis by checkpoint suggested an improvement in the average scores. The reason for this may lie in a mistake in plotting the SSIM curves in the notebook.

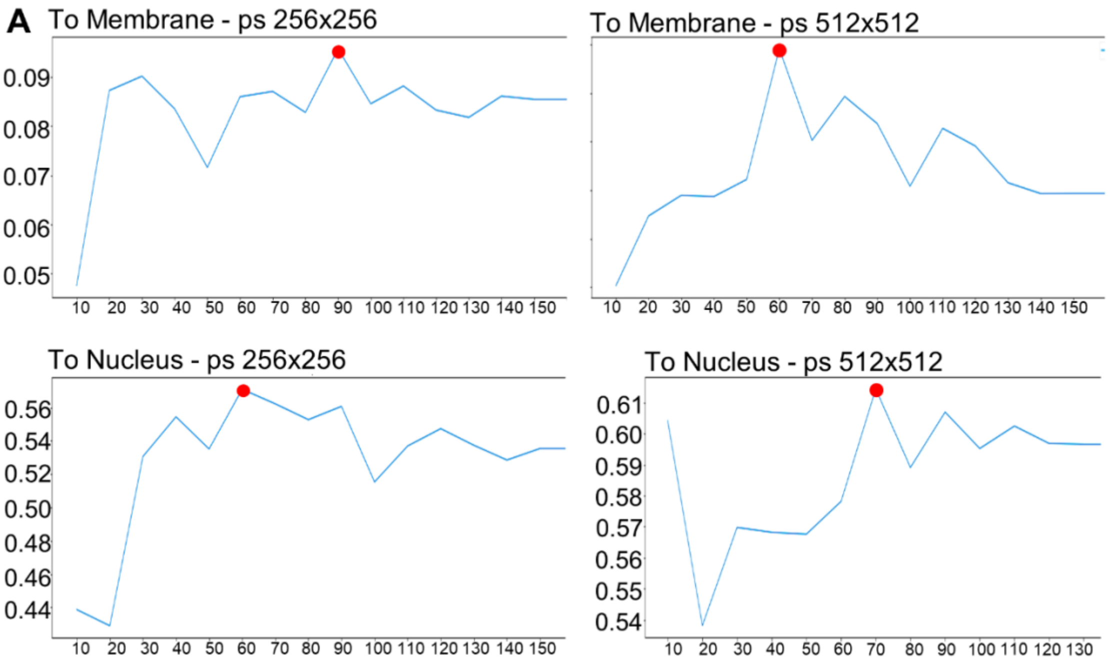


***Figure 51A - Finetuning pix2pix models for membrane-nucleus unmixing by patch size** – SSIM by training checkpoint for each model. The red spot indicates the model checkpoint used for evaluation. The model curve shown in the lower right was trained for 150 epochs but the first two checkpoints the notebook disconnected in the first runtime, the SSIM for checkpoints were not higher than the ones shown in the figure.*

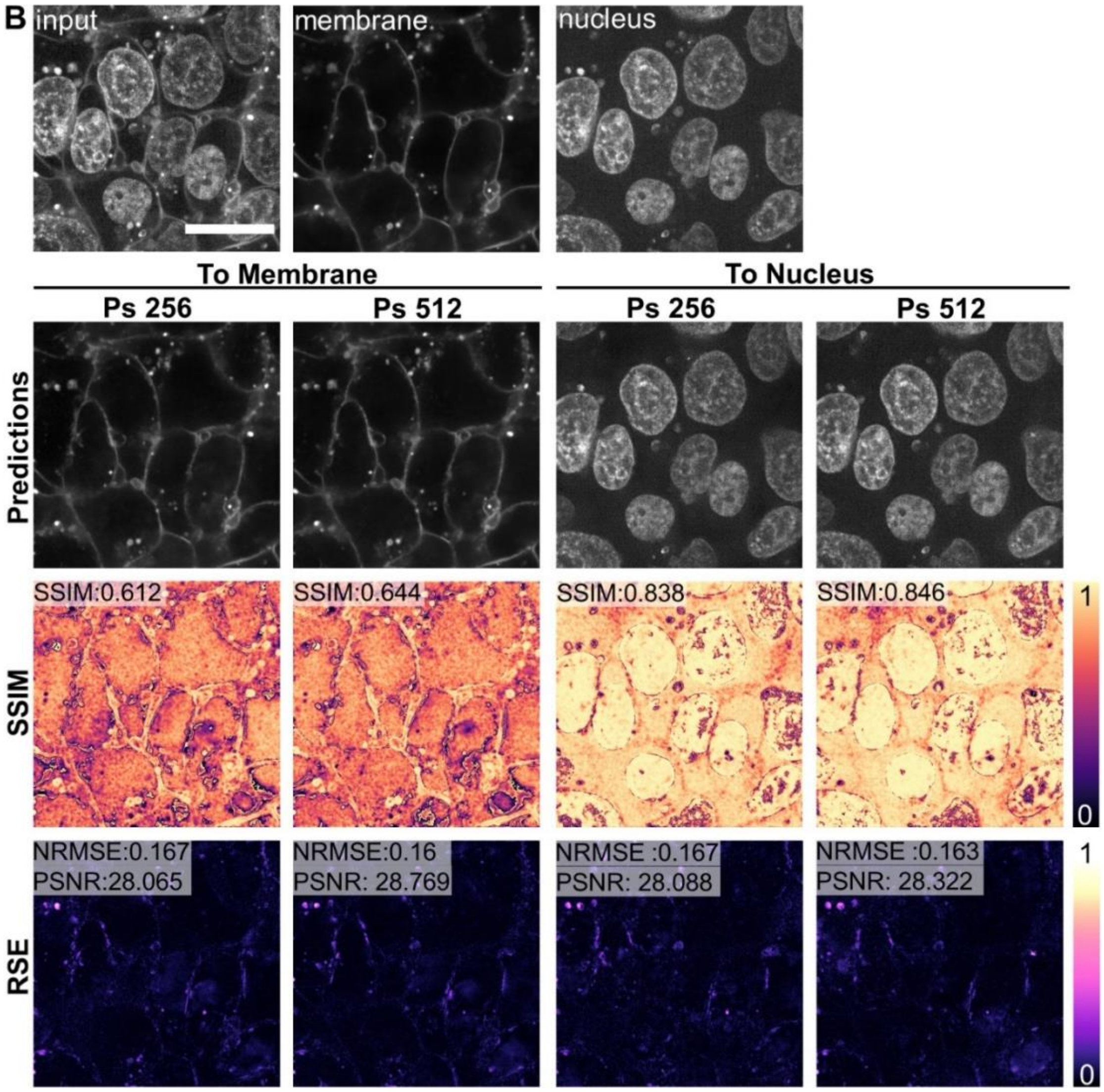


Figure 51B - Finetuning pix2pix models for membrane-nucleus unmixing by patch size *– Comparison of predictions of pix2pix before and after training with different patch sizes in the ZeroCostDL4Mic notebooks – Top: composite input (nucleus + membrane) and targets (nucleus, membrane); Middle panel: Predictions of membrane channel (first and second columns) and nuclear channel (third and fourth column) of pix2pix models trained with a patch size of 256x256 (first and third column) and pix2pix models trained with a patch size of 512x512 (second and fourth column). (scale: 20µm)*

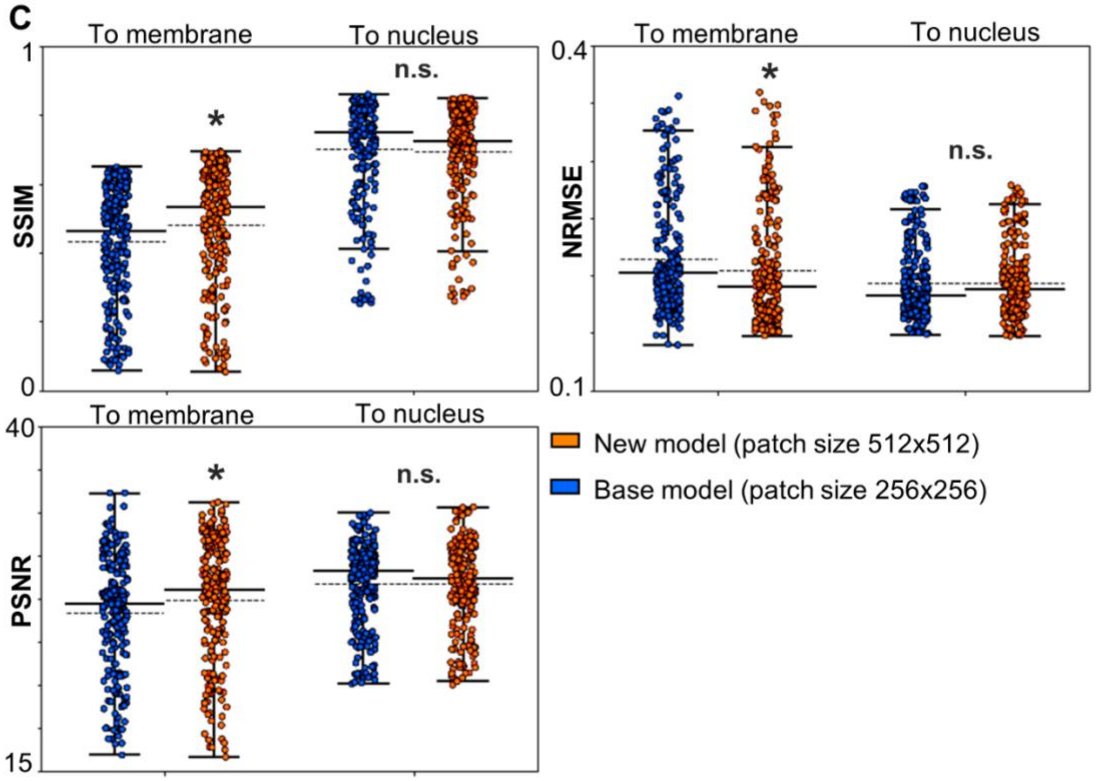


***Figure 51C - Finetuning pix2pix models for membrane-nucleus unmixing by patch size** – QC metrics for the full test dataset for the models above. (The whiskers encompass the range between the first and third quartiles of the distributions, with the median shown as solid, and the mean as a dotted black line). The asterisks indicate where the distributions differ significantly (with p<0.01, according to Mann Whitney U-test) from the base model.*

IV. 4. Results Chapter 3 - Summary

In this chapter, I designed a task in which the potential of the ZeroCostDL4Mic platform could be demonstrated. In this task, models were trained for a type of spectral unmixing challenge, to filter out labels from a composite image. Here, this was done as proof-of-concept on a dataset which was constructed from the individual channels in a public 3D dataset adapted for this chapter. In the first part of this chapter, I showed the results of three different methods, CARE, fnet and pix2pix, trained on this task. Comparing the models using the outputs from the ZeroCostDL4Mic QC section showed that all models learned to perform the unmixing tasks, with fnet performing better than CARE and CARE better than pix2pix, although pix2pix could only be trained on an 8-bit instead of a 16-bit dataset, which could compound the results. There were also clear differences in performance for different labels to be unmixed, with nuclei generally better recovered than membrane or mitochondrial labels from a three-channel image. When models were trained to separate 2 channels, the models performed best for unmixing nuclei from membranes and worst for unmixing membranes from nuclei and membranes from mitochondria.

The models perform better on those images with a lot of structures present in the image and worst on images with out-of-focus light, suggesting the performance does not depend on models learning to translate one blurry image into another.

The differences between the models and datasets were most easily identified on the error maps constructed in the QC section and the analysis of the distributions of the QC metrics on the full dataset, whereas the predictions themselves often appeared extremely similar and were not suitable for model comparisons.

In the second section, I tested how much the performance of some models could be improved given that the initial parameter changes may not have been suited for all models. To simplify this task, I focused on one of the unmixing tasks, from a combined nucleus+membrane composite to the constituent channels. Parameter tuning indeed led to some improvements, specifically in the reconstruction of the membrane channel. These improvements were more pronounced in the 3D models than in the 2D models. However, the improvements tended to be small and, in some cases, not statistically significant.

IV. 5. Discussion

The ZeroCostDL4Mic platform is intended to be used for custom tasks designed by users, and thus needs to provide a range of different tools to find a suitable solution for users. Here, an unmixing task was designed that could easily find applications in bioimaging, as phototoxicity caused by inducing fluorescence, the effects of bleed-through, the availability of imaging channels, and their acquisition are all problems that persist in many microscopic imaging studies. Using a DL approach instead of a physical filter or different wavelengths for imaging would be an attractive solution to such imaging challenges. In this chapter, I endeavoured to demonstrate that the platform created in this project can be used for such an image analysis task with relative ease and provides enough information for users to make informed choices about the applicability of DL for these tasks.

The key aspect to determine in this task is if the effort of using a ZeroCostDL4Mic implemented DL-approach is worth the potential performance achieved by the trained models. Therefore, two questions must be considered when assessing the feasibility of this approach. Firstly, can models reach the desired performance on this task? Secondly, are the time and effort invested in training the models justified by their performance?

In this chapter, three different methods, CARE, fnet and pix2pix were tested for the task of unmixing fluorescent channels from composite images. None of these methods is designed for this task but the tasks they were designed for fulfil very similar purposes to the one suggested here: Denoising models like CARE must learn to distinguish structures of interest from background, Label-free predictions (fnet) learn to filter out structures of interest from an input, and pix2pix should be able to learn nearly any image-to-image translation.

The results of this chapter suggest that all methods can be successfully trained for this task, although with some caveats and to different degrees for different cellular structures. Unsurprisingly, the models generally performed better when fewer labels needed to be removed from the composite image. Hence, images with two channels were unmixed more successfully than images with three channels. However, there were significant differences in performance for different structures in the images. For example, filtering the nucleus from a three-channel image was achieved with a better performance according to the error metrics employed in the notebooks than filtering either membrane or mitochondrial channels from any two-channel image. The performance peaks were reached for images that showed in-focus structures and were lowest for blurry or out-of-focus images. For use-cases of the unmixing approach this is an encouraging result as this suggests that for images which are of most biological interest the models will predict close to their peak performance. For the structures which were unmixed well, this means an average SSIM of between 0.7 or 0.8, for tasks which involve the nucleus or mitochondria. The performance of the models also increases primarily for images which are within focus. This was shown for each type of model using only one or two parameter changes in the notebooks. Any improvements tended to be subtle and led only to a small increase in quality, as measured by the quality metrics. However, the parameter search was not exhaustive in this chapter, and it is likely that users could further improve their DL-models if further parameters are adjusted, with each optimisation step having the potential to increment performance further. However, it will need to be determined if such incremental improvements are of benefit to the use-cases envisioned by the user. This leads to a second question, is the effort in training these models is justified by the performance? There are two reasons why this question could be answered positively. Although other tools for unmixing exist, the envisioned application of acquiring multiple labels in the same channel on purpose and using software for unmixing is well suited for a DL task. In the case of this chapter, unmixing was even performed without knowing the emission spectra of the labels, which would have already made linear unmixing difficult. This makes DL a less biased approach for this task than linear unmixing. Furthermore, linear unmixing tools may not even be easier to use than notebooks. Multiple different fiji plugins for linear spectral unmixing[211,212] were tested to give a better insight into the performance of the approach used in this chapter. Unfortunately, none of them yielded relevant results due to difficulties in use and potential compatibility or version issues which could not be resolved by the time of writing. It should be noted however, that even with proper use, these tools require significantly more user input than the method proposed here. Both require hand-labeled reference areas for each channel and the annotation of multiple individual images. This means that the relative effort in training models in ZeroCostDL4Mic could be smaller than using the standard unmixing tools. However, the ZeroCostDL4Mic approach has the disadvantage that it is currently not integrated into tools such as fiji, such as the LUMoS tool developed by McRae et al.[210] Unfortunately, the tool could also not be made to work in fiji for the work in this chapter to compare to the results of the methods implemented in ZeroCostDL4Mic. In fact, the option of using unsupervised learning for unmixing tasks is extremely attractive compared to the relatively long training process required for the models presented in this chapter. Especially, the performance of the best model, fnet 3D, for this specific dataset came at the cost of a significantly longer training than the other models. However, supervised learning may have a slightly lower risk in identifying incorrect labels as it is strongly trained on the structures of interest. It would have been interesting to compare the performance of the models in this chapter with the LUMoS method to test this hypothesis.

Another benefit of the use of ZeroCostDL4Mic is the strong emphasis on QC which in this chapter led to a rich source of information about the models and the datasets that were readily accessible when comparing the models and the different tasks. Being able to use QC on this task was crucial since the predictions of all models appeared extremely similar and could have easily led to the assumption that certain unmixing tasks were well solved although they were not.

Conversely, the QC section also indicated those unmixing tasks which may be most suited for exploitation in simultaneous acquisitions of labels, such as nucleus from different composite images or mitochondria from the membrane channel. The detailed results provided during QC could inform imaging experiments in which these channels could be imaged simultaneously in one channel with a relatively limited loss of information. It also allows users to weigh the risk of this approach, including the introduction of artefacts and misidentification of labels, against its benefits, which includes the potential for fewer acquisitions in multilabel-experiments or the potential for additional imaging channels. However, it should be stressed that even if the use of DL introduced artefacts into the images, the potential to acquire additional channels could outweigh this by increasing the content of information which could be acquired in an imaging experiment. In this way, DL may indeed aid instead of impeding the acquisition of new knowledge.

The results of this chapter itself offer some potentially intriguing insights into the DL tools used here and the datasets on which they were applied. Since the unmixing task shares features of denoising and label-prediction tasks, it is ideally suited to investigate the performance of different methods against one another using an equivalent experimental set-up. Overall, the fnet models perform better than the CARE models under the conditions tested in this chapter. Furthermore, both fnet and CARE models performed better than the pix2pix method, a comparison which is compounded by the different bit-depth of the image data used in the pix2pix model. Within the CARE and fnet models, both methods performed better in their 3D than in their 2D implementation even when the total size of the batches in terms of pixels remained the same. Differences between the performance of the 3D and 2D models could be visually detected and was also clearly visualised using the quality control metrics and error maps in the notebooks. The performance improvement upon increasing the patch dimensions in z has been reported for both fnet and CARE for their original tasks, and thus the performance improvement may not be surprising in this related task. From the experiment in CARE 3D, it further appears that deeper z-stacks in the image patches lead to better performances of the trained model. Interestingly, a reduction of the patch depth to half the depth of the full stack led to a significant decrease in performance, even below the level of the 2D model. This may indicate an issue with the dataset, as some stacks may contain several slices with little structural information. If the smaller patch depth leads to many images with little structural information in the stack, perhaps this makes it more difficult for the model to learn the task. That the 2D network performs better may then simply be a result of the larger total number of examples available to the network which may compensate the effect of blurry images to some degree.

In another line of questioning, the results from this chapter may also give insights into the data itself. Firstly, it is notable how the ranking of the models by task is identical across all models. For instance, it appears to be easier to unmix the membrane of mitochondria than the membrane of nuclei, regardless of the method used for the task. This suggests that there are inherent features in the data that make certain tasks easier to perform than others, independently of the method used to perform the task. The above example, where the membrane (membrane) label was more readily unmixed from the mitochondrial label than the nucleus, is interesting because it is not intuitive and even appears contradictory to the results from the task of recovering the nuclear signal, which all models perform relatively well in. It can be speculated that the structure of mitochondria and their localisation in both 2D and 3D images is not as easily confused with the pattern of the membrane labels as a human observer might predict. The fact that this task is easier to perform for the DL than recovering nuclei from a composite including a membrane or mitochondria may also be due to the relative size of the nuclei in the images. It could be speculated that it is inherently more difficult to remove a larger number of pixels correctly than the relatively small number of pixels associated with the membrane and mitochondrial structure. However, this is likely not the full reason for these observations and indeed, overlap is a further compounding factor. In this experiment labels were chosen that largely occupied exclusive regions of the cells (nucleus, cytoplasm, and membrane) and even in this context some unmixing tasks appeared more challenging than others. In further experiments with other labels, it could be tested to which degree overlap may influence unmixing performance. Potentially, the unmixing task could then be used as a pretext task to identify relationships between different cellular compartments or in different conditions. For example, a DL model trained on a control dataset (e.g. cells from a wildtype model) dataset unmixing two labels may identify subtle changes in the labels upon genetic or chemical manipulations.

Although the methods perform relatively well on the task designed for this experiment and could potentially be used in a further downstream analysis of the DL methods themselves or the datasets, there are also some practical caveats in the approach. The task developed for this chapter is similar to a multi-class semantic segmentation challenge where different areas of an image must be distinguished by the algorithm and separated for example by drawing outlines around different classes. Methods for performing such tasks already exist[166,213,214] and give insight into how the presented method could be improved. For instance, unlike a multi-class segmentation approach or even the LUMoS method[210], the models trained here can only output a single channel from the input image which means for a decomposition of input images into multiple different channels several models are necessary which may be unpractical for users. It would simplify the challenge for example, if a single model could be trained with a composite input image with all colour channels mixed and an RGB target image where each channel is given only one colour. While this may indeed be a more practical approach, it would also require more sophisticated quality control methods, as each channel would need to be compared to its target. The design of such quality control methods for this project was not feasible in the given timeframe. Another potential avenue to simplify this task would be a training approach where, instead of training a model with image pairs, models are trained with an input image and a target multiplet of images. The trained model could then output multiple images instead of one, and images could also be predicted in grayscale with a higher bit-depth than used in RGB images. These approaches would be desirable for the development of dedicated tools for the task designed here.

Nonetheless, this chapter shows that the methods already implemented in the ZeroCostDL4Mic project can be used to replicate tasks such as multi-class segmentation, albeit in a simplified form, and providing users with an easy-to-use avenue for testing the feasibility of such approaches for their datasets. Indeed, in publication of the tool, we already showed that a DL task, for cell tracking without the use of nuclear labels, which was published elsewhere without our previous knowledge could be replicated in the ZeroCostDL4Mic notebooks with relative ease-of-use. The task designed for this chapter is another confirmation that there may be other DL approaches which can be replicated with tools implemented in the ZeroCostDLMic platform. Despite some of the limitations of the approaches identified here, the ease of use, the rich output of information in the QC and the versatility of the tools in this platform are all indications that some of the key aims set out in the first chapter were achieved. The DL tools were run entirely in a browser that can be replicated anywhere in the world with Internet access (1. Dissemination problem, 3. Hardware problem). The methods were easy in use and the workflow was easy to replicate between all models, with input paths and output path seamlessly copied and pasted between notebooks, facilitating the use of all DL tools (2. Knowledge Problem). The QC gave detailed insights into the models’ performance and ultimately the datasets themselves and potentially informs further imaging experiments and model finetuning. All these observations suggest that the ZeroCostDL4Mic method developed in this project did achieve its core objectives and could provide a valuable contribution to the democratisation of DL in microscopy.

IV. 6. Methods

Creating the datasets

The dataset used in this chapter was taken from the Allen Institute for Cell Science's microscopy pipeline (see http://www.allencell.org). The dataset is the same as that used in the Label-free prediction method published by Ounkomol et al. The images show human embryonic kidney cells (HEK293), human fibrosarcoma cells (HT-1080) human induced pluripotent stem-cells (hiPSCs) with endogenously mEGFP tagged cellular structures: DNA for nuclear labelling, TOM20 for mitochondrial labelling and CAAX-tagged mTagRFP for membrane labelling.

After choosing the labels, the stacks of interest were cropped from this dataset with dimensions of 512x512x32 and converted from the Allen Institute custom format into the more Colab compatible .tif format using the fiji bioformat importer and saving the images in fiji. The individual channels were converted to grayscale and normalised to cover the full dynamic range of 16-bit depth, using the enhance contrast tool with thresholding for the top 0.3% of pixel intensities. To create the merged channels, the individual stacks were merged using the fiji Math tool and adding image channels to one another, in the following combinations for the input datasets:

Nucleus + membrane + mitochondria, nucleus + mitochondria, nucleus + membrane, mitochondria + membrane.

The target datasets consisted only of the individual channels, nucleus, membrane and mitochondria.

For the 2D methods, the stacks were split along their z-axis into individual slices. For the pix2pix method these slices were converted to .png format in fiji.

The datasets used to train the models in this chapter are accessible in this Google Drive folder:

https://drive.google.com/drive/folders/1QNJFXul-A69PEFgVfRKxO1L2MJzz3Q6L?usp=sharing

Training

All models were trained on the above datasets with the respective Colab notebooks provided in the ZeroCostDL4Mic project. The GPUs assigned for training were the

Tesla K80, Tesla T4 and Tesla P100C. The parameters were chosen within the notebooks, with the exception of fnet where the patch size was changed by editing the training.py file within the Colab environment. The learning rates were not changed in the notebooks from their default settings in the notebooks, with 0.004 for CARE, 0.002 for fnet and 0.001 for pix2pix, with CARE and pix2pix both using learning rate decay (although this was never triggered in the CARE models in chapter).

Because training could take several hours and Colab allows only one simultaneous session, to increase the number of models trained at once, multiple Google accounts were used simultaneously. The results path was set to a folder shared between all google accounts so that models were accessible from a single google drive folder.

Quality Control

QC in the notebooks was performed using the tools implemented within the notebook. To accelerate this and reduce the memory footprint for pix2pix, lpips analysis was disabled for this notebook. The example images in figures 41 to 46 and 49 to 51 were chosen after visually assessing the ground-truth stacks in the test dataset for high contrast and in focus structures. The same image was chosen for the figures in part I.2. (14th slice of stack 72) and another for the figures in I.3 (13th slice of stack 74) to allow the performance for the tasks to be more easily compared. For the 2D models, the images and error maps were downloaded from the google drive manually after QC, and normalised in fiji, the error maps converted using the mpl-magma LUT in fiji. For the 3D stacks, these steps were identical, but the relevant slices were extracted from the stacks in fiji.

Losses were recreated from the loss .csv files of the models for the fnet and CARE models. For pix2pix, the raw .png files exported by the notebook were used to display the checkpoints and SSIM scores as the scores were not saved in a separate document.

The plots for quantitative analysis of the models on the full datasets were performed using the *seaborn.stripplot* and *matplotlib.boxplot* libraries. A script was written in a Colab notebook to extract the values for the quality metrics from the .csv file which is saved at the end of the QC step in the notebooks. The statistical analysis was carried out within the same script. Since the distributions were not assumed to be normal and were skewed towards higher scores, the distributions were compared using the nonparametric two-tailed Mann-Whitney U test, implemented using the SciPy[175] stats module.

V. Summary and Discussion

“Deep Learning” has become a buzzword in the community of biologists and bioimage analysts as a novel technology offering many solutions to problems which researchers in these fields are faced with. This includes the potential to reduce phototoxicity, one of the central remaining problems in live-cell imaging, by training DL-models to predict accurate high SNR images or to detect signals in images taken with live-cell friendly acquisitions, e.g. brightfield. On the analysis side, DL tools allow more complex data to be extracted and analysed from very large datasets some of which were not possible or feasible with many previously existing technologies. Yet, when attempting to use the ground-breaking potential of DL at the outset of this project, it became evident that many of the tools available for the bioimaging community were surprisingly difficult to wield, specifically for a novice user. Initially, the project was motivated simply by the need to create a DL pipeline that was simple in use, and importantly one that allowed models to be trained on custom datasets. The latter point was important because several DL tools published for bioimage analysis tasks such as classification did not provide clear avenues for users to train these models. Instead, authors often recommended using their trained models for analysis[147,178,183,187] or give mostly instructions on how to reiterate the training done for the publication on datasets provided with the paper[114]. However, for microscopists aiming for example to reduce phototoxicity in their imaging pipelines by denoising, pretrained models are not necessarily a robust option because their ability to generalise to a user’s data cannot be guaranteed, something that was specifically shown in the ZeroCostDL4Mic paper[134]. Similarly, detection or segmentation of images will almost always improve upon training of models on the required data. Hence, the greatest benefit of DL to users in bioimaging comes from training models on their own data. This is the key aspect of DL that this project needed to facilitate. Initially, this meant identifying what obstacles prevented me (and potentially many users with similarly limited background in computer vision and coding) from training DL tools for imaging experiments and analysis. These are the five problems outlined in the introductory chapter of this work: 1. Dissemination problem, 2. Knowledge problem, 3. Hardware problem, 4. Reliability Problem and 5. Dataset problem.

V. 1. Chapter Summaries

V. 1. 1. Results Chapter 1

The main aim of this project was then to address these problems which led to the creation of the ZeroCostDL4Mic platform in collaboration with Guillaume Jacquemet and Romain Laine which is in essence a repository of Google Colab notebooks. The availability of these notebooks through a web browser is an immediate solution to the first problem which is concerned with bringing DL into the hands of novice users. Since most researchers today have at least access to the web via a browser, the notebooks are more accessible than many other tools which require downloading software packages or repositories, and the installation of dependencies, beginning often with Python which many biologists have never used. At the same time, the use of Colab was attractive because it came with limited but free access to GPUs, with the necessary software to use TensorFlow already integrated by default. Creating a platform around this free resource was key to achieving the goal of democratising DL, specifically solving the abovementioned 'hardware problem'.

Once the central resource for the envisioned platform was chosen and the crucial obstacle of GPU access resolved (albeit with limitations), the other key problems could be addressed. The tool needed to solve the problem that many potential users would not know how to use a DL-tool, which provided a challenge in designing an intuitive workflow users could easily follow, but also an opportunity, as this tool could be didactic and as an entry tool could encourage good-practice of DL. This led to the workflow introduced in the first chapter of this work, containing *the four steps* which is followed in all notebooks, including those steps which are necessary for the use of DL in Google Colab. The implementation of a consistent workflow was a key objective in solving the second problem (knowledge problem) as it should help to build familiarity and intuition with different DL-tools, which ultimately follow the same basic steps. The final consideration which informed the design of the ZeroCostDL4Mic notebooks was that without exception all DL-methods required coding or code interaction for models to be trained on custom datasets by users. If not necessarily a practical, this may yet represent a psychological barrier for some users. Inspired by widely used tools in bioimaging such as ImageJ/Fiji[53,215], it was seen as a highly desirable feature for the democratisation of the platform if it could be used, at least by the most novice users, completely without code-interaction via a GUI. Indeed, the Colab environment, unlike standard Jupyter notebooks, allowed this to be achieved via the so-called Colab forms module. Using this module, the entire workflow could be implemented without the user having to read or interact with code. Instead, all parameters and paths could be implemented with interactive tools. In this aspect, the work is one of the first to provide code-free access to DL, although DeepImageJ[216] has achieved a similar goal through some applications running within the ImageJ GUI. This has the advantage that many bioimage analysts will already be familiar with the GUI and will benefit from the highly active community using and improving Fiji and ImageJ. However, crucially, and unlike DeepImageJ, the ZeroCostDL4Mic platform does not require users to have GPU access to train models. Furthermore, the availability of source-code in the Colab notebook makes adaptations of the code by users with some experience in coding easier to perform and test than in an ImageJ or Fiji tool. The ZeroCostDL4Mic workflow and notebook layout also invites users or developers to implement their tools or workflows within this format. Implementing new tools in DeepImageJ[216] will likely not be as simple for many users and even developers as it requires their methods to be cross-compatible with Java, a complicating aspect since most DL is developed in Python.

This project's solution to code-free interaction with DL meant that differences in the source code of different methods could be hidden from the user behind a consistent GUI, which was crucial in creating the desired workflow and intuitive layout of the notebooks. However, combining the goals of a consistent workflow and the interactivity of the notebooks was perhaps the most significant challenge encountered in this project because it required converting vastly different DL solutions from different authors into a single format. While challenging, it allowed me and the other developers of the ZeroCostDL4Mic platform to create a tool which would combine the best elements of these tools while improving potential shortcomings. Here, it became clear that the tools created by the CSBDeep project, specifically CARE[113] and StarDist[147,187], were already designed relatively intuitively and well documented, making them easier to include in the ZeroCostDL4Mic platform. Specifically, the implementation of CARE in the ZeroCostDL4Mic notebook thus acted as guidance for some other methods which were more challenging in implementation, specifically YOLOv2[94] and fnet[114].

Although the creation of the ZeroCostDL4Mic platform is the chief outcome documented in chapter one, the detailed review of the various methods chosen for implementation also revealed many practical hurdles which contribute to the second problem (knowledge problem). Identifying these issues could be of interest to projects beyond the presented one as it could act as simple guidance for developers to make their DL-tools more suitable for the purposes of bioimage analysis and microscopy. For instance, most methods do not by default provide the user with a readable documentation of a DL model's training, instead providing .log files or simply Python history objects which may not always be human-readable. In ZeroCostDL4Mic, this problem was solved by having all notebooks output a summary pdf document which contains the most important information about the training of the model, including hyperparameters, loss curves and details on the datasets used in training. Another issue common for different DL methods is that they are not agnostic to file-type. This is an issue which is specifically difficult to overcome for bioimaging where many different image file-types are used. Surprisingly, most methods are not designed to accommodate different filetypes but are instead designed to accept either tifs, .pngs or custom filetypes such as .czi (fnet) for image files. This problem could not be solved within the scope of this project as it would have required thorough changes in the source-codes of each method which was not the aim of this project. However, it is an example of a relatively simple shortcoming in many methods encountered in this project the removal of which would make it significantly easier to use these tools. Since all tools ultimately create NumPy[174] arrays to create the training tensors, it should not be difficult for developers to make their tools agnostic to filetype by converting various file-types into this format within their tools.

Although all the methods present in the project were successfully implemented on the ZeroCostDL4Mic platform, some shortcomings remained in the fulfilment of a key goal of the project, namely consistency. Though the layout of the notebooks is identical it was not achieved to reach perfect agreement between the notebooks. This was partly due to inherent differences in the methods, e.g. YOLOv2 which uses completely different filetypes than the purely image-based methods, partly due to convenience, i.e. the retraining section in fnet is specifically designed to accommodate time-outs rather than transfer learning, and partly due to the collaborative nature of the platform's creation. The latter point should be highlighted here as the project contained many more methods than the ones presented in the three chapters of this work, which were those I was primarily concerned with. With each notebook being designed by a different developer, achieving consistency, e.g. in versioning, was a challenge which was met by regularly copy-pasting common text and code-fragments to each notebook. Yet, some small inconsistencies may still occur in some notebooks, e.g. in type-cases and variable names. Further adjustments will be necessary but also increasingly challenging as more notebooks are added to the repository to achieve the goal of full consistency.

Another limitation some users may encounter is the limited choice of parameters for some of the notebooks. For instance, none of the notebooks allow adjustments to the kernel size of the convolutions of the neural networks even though this is likely to affect performance of the models during training. Other hyperparameters such as depth, the loss function, dropout rate and similar commonly adjusted parameters were not implemented as parameter choices in most notebooks. The main reason for this was that in this way the methods stayed closer to their originally published versions. Changing the patch size or learning rates (which are implemented in most notebooks) did not change the fundamental working of the published algorithm and were therefore included. The other reason is that adding additional parameters could confuse first-time users. Since the networks can be trained to fit almost all the challenges they were presented with in this work, the tuning of additional parameters is not necessary for novice users to understand and train a DL-model. However, the obvious disadvantage of this choice is that certain hyperparameters and the associated benefits in tuning them can only be accessed using the code within the notebooks and not through the GUI.

V. 1. 2. Results Chapter 2

After the core platform was created and several tools could be trained and used for prediction, the fourth problem, reliability, became of interest, as none of the models' predictions could be inspected for errors except by eye. Here, it became evident how little attention many methods paid to quality control. Few methods even included inbuilt quality controls despite all methods being at least partially quality controlled in publication. This is an unexpected shortcoming since quality controls are crucial to the reliability of DL tools and because their implementation is not necessarily more difficulty than building the DL tools themselves.

Hence, it became a significant additional goal in this project to design an intuitive and powerful quality control (QC) section in which any model could be assessed on a test dataset and to include this as part of the core workflow in all the notebooks. In the ZeroCostDL4Mic QC section, the core metrics chosen were based on the existing literature, i.e. those metrics which were used in the majority of the methods in publication. This led to the use of the NRMSE, PSNR and SSIM metrics as the core metrics for image-to-image tasks and IoU, AP, F1 and (for StarDist) panoptic quality, for the instance and semantic segmentation and object detection tasks. Using QC is essential to detect problems and rigorously improve DL models. Hence, it needed to be demonstrated that the QC section could show that models in the ZeroCostDL4Mic platform could be trained to good performance on datasets expected users might intend to use DL tasks for, often relatively small datasets. Furthermore, the metrics implemented in the notebooks should be sensitive enough to register small improvements or deterioration of models and translate into useful information to the user. Indeed, the QC metrics provided enough evidence to suggest that models trained on the ZeroCostDL4Mic platform could achieve good performance even when trained on relatively small datasets. The analysis of the methods was at times hampered by the limited number of test images which made it difficult to assess significance in some of the tasks. Furthermore, all models could be fine-tuned to some degree via parameters available within the notebooks, which means users can improve their models until satisfied with the results, an important aspect in DL which is notoriously empirical.

In implementing these metrics, a crucial observation was made regarding commonly used image-to-image quality metrics, which influenced the implementation of QC in the notebooks, and which has so far hardly been discussed in the literature. Firstly, the implemented image metrics alone give very little information about the performance of a model on an image as these are image-wide metrics and do not resolve problematic image areas, artefacts, or potential biases all of which would help understand how well the model performs. On individual images the metrics are therefore nearly unusable, unlike the much more intuitive IoU, AP and F1 metrics and p-r curves which provide a clearer picture to a user about the performance of a model on an image and full image set. There are two likely reasons for this discrepancy, first, the nature of the tasks themselves. Classification tasks are inherently easier to score than regression tasks, such as image-to-image analysis where a ‘perfect’ prediction, i.e. identity with the ground-truth, is likely impossible except in trivial tasks. This means there is more room for interpretation by the user as to what constitutes are desirable performance of the model and makes it more difficult to score. The other reason may be that the field of image-to-image tasks in DL is comparatively small and is itself primarily concerned with segmentation tasks. This means the motivation to develop quality controls for tasks such as denoising or label-prediction in bioimaging, will be limited compared to classification and even segmentation tasks. Growing the user-base of DL tools and finding novel applications and problems which DL can solve could also increase the interest in creating more robust quality metrics for challenging-to-score tasks in the future. This may be another speculative benefit of democratising the use of these methods.

As yet such QC metrics do not exist, in this project an alternative solution was designed to assess image-to-image tasks. Instead of scoring models through quality metrics alone, the user can assess shortcomings of the models through error maps which resolve the differences between ground-truth and prediction in a test image pixel-wise. These error maps indeed provided a clearer picture for models’ performances and revealed, for example, a bias in CARE 2D for images with more background, which reached much higher scores than images containing proportionately more structure of interest. This means that image-to-image metrics widely used in the field need to be treated with caution if the test datasets are not adjusted for background and foreground structure. In practice, this can be difficult as this distinction could introduce bias, e.g. by allocating putative foreground-pixels to background and vice-versa. The solution provided in the notebook, i.e. with a human control, is currently the only feasible solution we found for reliable QC for image-to-image tasks.

The observations in this chapter suggest that comparisons of the same method on different structures which is common in the literature (e.g. CARE and fnet) may not be justified unless the test images are adjusted for background and foreground proportions. Further research in quality control of image-to-image tasks, specifically bioimaging tasks which differ significantly from similar tasks on macroscopic objects, needs to address the biases which can be introduced via foreground or background-heavy datasets.

V. 1. 3. Results Chapter 3

To test that the platform fulfils its intended purpose, in the final chapter, a novel task with motivation from microscopy was designed as a showcase of ZeroCostDL4Mic. Indeed, applications beyond the core tasks implemented in this project were already shown in publication in the form of combining two notebooks for cell-tracking, and in a recent publication by a collaborator to use DL on bacterial images. However, none of these have attempted to showcase a novel task for DL in bioimaging and that it is possible to achieve this in a rigorous manner through ZeroCostDL4Mic. In chapter three, I show that with the tools available in the repository, multiple different structures can be identified and unmixed from a composite image with tools developed for other purposes. In the process of this showcase, the platform can be shown to be used firstly to train all models on a novel task successfully, to compare the trained models to identify the best candidate for this task, and even to draw preliminary conclusions on some biological or image-based features in the dataset. Thus, the third chapter shows that the tool can indeed be applied for novel tasks and in a versatile manner. It also revealed some significant challenges in using the tool which could be of concern for its potential of democratising DL for microscopy. Of primary concern is the access given to GPU time by Google. In performing the trials in Chapter 3, timeouts were more frequently encountered than at the beginning of the project, likely because Colab has a growing user base that does not scale with the amount of GPU-time Google provides. This means that while providing a powerful resource to democratise DL, Google also acts as a gatekeeper to access this tool. This is also true for the storage space on Google Drive. Fortunately, in this project, I was able to access a drive account purchased for research purposes that has unlimited drive space. However, for standard users, even the use of more than one pix2pix[217] or cycleGAN[218] model with a full quality control of a medium-sized test set, 100s of images, could be difficult to store with standard Google Drive access. This can be mitigated by using the ability to share folders on the Google Drive between users, only one of whom may need to purchase additional space on the Google Drive.

There are different potential solutions to these issues. Firstly, users can purchase Colab Pro accounts which reliably give access to the fastest GPUs although this means that the project no longer is ‘zero-cost’ for users. However, it also means losing none of the benefits of using Colab. The second option is exploring other free-use GPU platforms as shown in the supplementary method of the ZeroCostDL4Mic publication although some of the GUI features will be lost. The third solution is to run the notebooks locally, storing files on local hard drives and training networks on local GPUs (https://research.google.com/colaboratory/local-runtimes.html). Although this solution may also not be zero-cost, it could become more feasible if GPUs become cheaper, which can be expected as demand and production of GPUs will likely continue to increase for years. This may also be a solution for more advanced users who may still want to use some of the shortcuts represented in the ZeroCostDL4Mic notebooks, such as loading dependencies, augmentations, QC steps, etc.

V. 2. Conclusions

Here, I have built one of the first GUI-based DL-tools in any research field, adapted for bioimaging. The tools have an intuitive design inspired by other popular GUI-tools such as ImageJ[53,215]. The tool is indeed simple in use and can be used quickly to train and compare many DL models. For chapters 2 and 3 alone over 500 models were trained, in less than a year. The project includes a quality control section which is currently not common in many DL workflows and automatically analyses models' performances with the relevant metrics allowing both quantitative and qualitative assessments of models. The project can be used for novel tasks and gives outputs which are insightful for both the analysis of the datasets and the DL methods themselves. The original goal of the project to use DL to address problems in microscopic imaging was shown in the final chapter to be by achieved by using the tool for task that could significantly reduce toxicity in imaging or be exploited to generate additional image data.

Although the access to Colab is important for the project, the merit of this project does not depend on resources provided by Google or other platforms. Several of the findings made throughout this project were made independently from Google's resources. The workflow itself and the solutions found in this project to provide code-free access to DL for users, could influence other tools which may be created in the future. By creating the ZeroCostDL4Mic repository, the community is provided with one potential solution to some of the issues encountered in the existing methods for DL in microscopy and bioimaging.

V. 3. Outlook

Already the development of high-level coding tools such as TensorFlow[191] and PyTorch[190] (and others) which can be easily used in Python and which has a dynamic and growing user-base has made a big impact on the democratisation of DL technology as it can be used or learned by users of Python, one of the largest coding languages in the world. Yet, it cannot be assumed that many potential beneficiaries of DL-tools in bioimaging are proficient in either DL or Python. This is the group this project intended to reach. As discussed in the introductory chapter, the extent of democratisation is not easily measured nor is the impact of a technology on science. However, since the publication of the project, several correspondences with users of the platform and the activity on the GitHub page suggest that there is interest in this tool from the community. Interest for the project has come from countries such as Brazil where resources for research are far more limited than in many European countries or the U.S. Open-access and free-to-use tools are likely of even greater impact in countries outside the academic hotspots of the world where students and researchers are less likely to have access to GPUs. This gives confidence that the tool indeed is reaching the targeted community and may perhaps inspire more users to test DL in their research.

There are also some applications which were not envisioned in the original publication of this paper but were developed by some early collaborators. The first of these is the DeepBacs[219] project which represents one of the first characterisations of bacterial image data using DL, and prominently used the ZeroCostDL4Mic resources. The format of the ZeroCostDL4Mic platform is also attractive for the implementation of further methods, and the project continues to grow, particularly owing to the efforts made by the Jacquemet group which also continues to maintain the existing notebooks. As further methods, including novel methods, are added, the project can likely remain relevant even as the initial notebooks may represent older methods which will be surpassed in the future.

However, there has remained an important gap in the achievements of the project which will need to be addressed in further research and which will be crucial for the further democratisation of DL in bioimaging and beyond. This gap lies in finding a satisfactory solution to the 5th problem identified in this project, concerning the need to curate or otherwise access large, paired datasets for training and testing of DL methods. In the ZeroCostDL4Mic project, several features were implemented in the notebooks to accommodate the potential lack of large training datasets in the form inbuilt data augmentation, patching, and the potential to re-train models via transfer learning on small datasets. However, these features do not solve the crucial issue inherent in the supervised methods which this project mostly implements (Noise2Void[220] and Deep-STORM[131] being exceptions), which is their reliance on large, paired datasets. Even the creation of relatively small datasets of up to 100 images was a significant undertaking, most significantly for those datasets which needed hand annotations, such as YOLOv2[94] and StarDist[187]. While StarDist performed well after training on a comparably small training dataset of only 28 images, in YOLOv2, it was already evident that the model essentially failed in its task if the training dataset of roughly 30 images were not augmented. The annotation problem is a widely known and discussed subject in DL and the associated bioimaging communities with several solutions being put forward in recent years. However, there remain some key challenges.

There are several potential avenues which can be explored to achieve this either in future versions of the ZeroCostDL4Mic notebooks or in a separate project. Firstly, can annotation be facilitated within the platform? This question could be cautiously answered with yes. In collaboration with Wei Ouyang, Romain Laine, one of the co-creators of the ZeroCostDL4Mic platform has created a Colab notebook, included in the ZeroCostDL4Mic repository which includes the kaibu[221] plugin which can be used to interactively view, edit and label images in the notebook. A similar web-based tool for segmentation tasks also exists in the form of the Deep Cell Label project which allows 2d, 3d and 4d segmentations of datasets[178]. Although such tools are useful as this means that, at least within the ZeroCostDL4Mic notebooks files required for training could be generated directly in the google drive rather than being uploaded, they do not solve the central problem that creating segmentation data manually is extremely time-intensive particularly for 3D segmentations. However, since tools such as kaibu[221] and napari[222] which also allows 3D segmentations are Python based, it is likely feasible to include some DL-powered tools into the segmentation pipeline, and this is already being done in early stages (https://colab.research.google.com/github/HenriquesLab/ZeroCostDL4Mic/blob/mast er/Colab_notebooks/Beta%20notebooks/ZeroCostDL4Mic_Interactive_annotations_Cellpose.ipynb). Such approaches might include user's inputs being used by the model to generate suggestions for further annotations which could make the annotation of a dataset significantly faster, or even lead to the creation of a trained model once a few images are fully annotated. In this way, it could be imagined that the user's input could be used directly to train a model on the fly that improves its predictions as the user adds more information to the annotations. Such tasks are already known in machine learning as 'active learning' and have hardly been explored in the context of image annotation. Other potential candidates for a DL-assisted labelling approach could be in simulating large training datasets if enough information is known about the task to create such a dataset, which could, for example, be done using a GAN. Indeed, this was attempted in the presented project but failed for unknown reasons and could not be solved within the timeframe of this project.

Ultimately, the supervised learning approaches are likely to be an intermediate step in the advances DL-tools will make for bioimaging and it is likely that many self-supervised or human-in the loop approaches will soon take over some of the tasks in this project. The ZeroCostDL4Mic project could be used as an inspiration for the intuitive implementation of such unsupervised methods in the future. Implementing such methods in an intuitive manner could be one of the next big innovations for bioimaging as it would offer a solution to the last crucial obstacle for the democratisation of DL in bioimaging.

VI. References

1. Esteva, A. *et al.* Dermatologist-level classification of skin cancer with deep neural networks. *Nature* **542**, 115–118 (2017).
2. Krizhevsky, A., Sutskever, I. & Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. in *Advances in Neural Information Processing Systems 25* (eds. Pereira, F., Burges, C. J. C., Bottou, L. & Weinberger, K. Q.) 1097–1105 (Curran Associates, Inc., 2012).
3. Maqueda, A. I., Loquercio, A., Gallego, G., García, N. & Scaramuzza, D. Event-Based Vision Meets Deep Learning on Steering Prediction for Self-Driving Cars. in 5419–5427 (2018).
4. Nugraha, B. T., Su, S.-F., & Fahmizal. Towards self-driving car using convolutional neural network and road lane detector. in *2017 2nd International Conference on Automation, Cognitive Science, Optics, Micro Electro--Mechanical System, and Information Technology (ICACOMIT)* 65–69 (IEEE, 2017). doi:10.1109/ICACOMIT.2017.8253388.
5. Rao, Q. & Frtunikj, J. Deep learning for self-driving cars: chances and challenges. in *Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems* 35–38 (Association for Computing Machinery, 2018). doi:10.1145/3194085.3194087.
6. Dubochet, J., McDowall, A. W., Menge, B., Schmid, E. N. & Lickfeld, K. G. Electron microscopy of frozen-hydrated bacteria. *J. Bacteriol.* (1983) doi:10.1128/jb.155.1.381-390.1983.
7. Adrian, M., Dubochet, J., Lepault, J. & McDowall, A. W. Cryo-electron microscopy of viruses. *Nature* **308**, 32–36 (1984).
8. Frank, J. *et al.* SPIDER and WEB: Processing and Visualization of Images in 3D Electron Microscopy and Related Fields. *J. Struct. Biol.* **116**, 190–199 (1996).
9. Yip, K. M., Fischer, N., Paknia, E., Chari, A. & Stark, H. Atomic-resolution protein structure determination by cryo-EM. *Nature* **587**, 157–161 (2020).
10. Nakane, T. *et al.* Single-particle cryo-EM at atomic resolution. *Nature* **587**, 152–156 (2020).

11. Chalfie, M., Tu, Y., Euskirchen, G., Ward, W. W. & Prasher, D. C. Green Fluorescent Protein as a Marker for Gene Expression. *Science* (1994) doi:10.1126/science.8303295.

12. Hell, S. W. & Wichmann, J. Breaking the diffraction resolution limit by stimulated emission: stimulated-emission-depletion fluorescence microscopy. *Opt. Lett.* **19**, 780–782 (1994).

13. Gustafsson, M. G. L. Surpassing the lateral resolution limit by a factor of two using structured illumination microscopy. *J. Microsc.* **198**, 82–87 (2000).

14. Betzig, E. *et al.* Imaging Intracellular Fluorescent Proteins at Nanometer Resolution. *Science* (2006) doi:10.1126/science.1127344.

15. Rust, M. J., Bates, M. & Zhuang, X. Stochastic optical reconstruction microscopy (STORM) provides sub-diffraction-limit image resolution. *Nat. Methods* **3**, 793–795 (2006).

16. Abbe, E. The Relation of Aperture and Power in the Microscope (continued).*. *J. R. Microsc. Soc.* **2**, 460–473 (1882).

17. Xu, K., Zhong, G. & Zhuang, X. Actin, Spectrin, and Associated Proteins Form a Periodic Cytoskeletal Structure in Axons. *Science* (2013) doi:10.1126/science.1232251.

18. Nguyen, J. N. T. & Harbison, A. M. Scanning Electron Microscopy Sample Preparation and Imaging. in *Molecular Profiling: Methods and Protocols* (ed. Espina, V.) 71–84 (Springer, 2017). doi:10.1007/978-1-4939-6990-6_5.

19. Wäldchen, S., Lehmann, J., Klein, T., van de Linde, S. & Sauer, M. Light-induced cell damage in live-cell super-resolution microscopy. *Sci. Rep.* **5**, 15348 (2015).

20. Tosheva, K. L., Yuan, Y., Matos Pereira, P., Culley, S. & Henriques, R. Between life and death: strategies to reduce phototoxicity in super-resolution microscopy. *J. Phys. Appl. Phys.* **53**, 163001 (2020).

21. Icha, J., Weber, M., Waters, J. C. & Norden, C. Phototoxicity in live fluorescence microscopy, and how to avoid it. *BioEssays* **39**, 1700003 (2017).

22. Frigault, M. M., Lacoste, J., Swift, J. L. & Brown, C. M. Live-cell microscopy - tips and tools. *J. Cell Sci.* **122**, 753–767 (2009).

23. Icha, J., Weber, M., Waters, J. C. & Norden, C. Phototoxicity in live fluorescence microscopy, and how to avoid it. *BioEssays* **39**, 1700003 (2017).

24. Brown, C. M. Fluorescence microscopy - avoiding the pitfalls. *J. Cell Sci.* **120**, 1703–1705 (2007).

25. Henriques, R., Griffiths, C., Hesper Rego, E. & Mhlanga, M. M. PALM and STORM: Unlocking live-cell super-resolution. *Biopolymers* **95**, 322–331 (2011).

26. Wegner, W., Mott, A. C., Grant, S. G. N., Steffens, H. & Willig, K. I. In vivo STED microscopy visualizes PSD95 sub-structures and morphological changes over several hours in the mouse visual cortex. *Sci. Rep.* **8**, 219 (2018).

27. Chen, B.-C. *et al.* Lattice light-sheet microscopy: Imaging molecules to embryos at high spatiotemporal resolution. *Science* (2014) doi:10.1126/science.1257998.

28. Diekmann, R. *et al.* Characterization of an industry-grade CMOS camera well suited for single molecule localization microscopy – high performance super-resolution at low cost. *Sci. Rep.* **7**, 14425 (2017).

29. Huff, J. The Airyscan detector from ZEISS: confocal imaging with improved signal-to-noise ratio and super-resolution. *Nat. Methods* **12**, i–ii (2015).

30. Wang, F. *et al.* Scanning superlens microscopy for non-invasive large field-of-view visible light nanoscale imaging. *Nat. Commun.* **7**, 13748 (2016).

31. Wang, Z. *et al.* Optical virtual imaging at 50 nm lateral resolution with a white-light nanoscope. *Nat. Commun.* **2**, 218 (2011).

32. Meijering, E., Carpenter, A. E., Peng, H., Hamprecht, F. A. & Olivo-Marin, J.-C. Imagining the future of bioimage analysis. *Nat. Biotechnol.* **34**, 1250–1255 (2016).

33. Wollman, R. & Stuurman, N. High throughput microscopy: from raw images to discoveries. *J. Cell Sci.* **120**, 3715–3722 (2007).

34. Quintavalle, M., Elia, L., Price, J. H., Heynen-Genel, S. & Courtneidge, S. A. A cell-based high-content screening assay reveals activators and inhibitors of cancer cell invasion. *Sci. Signal.* **4**, ra49 (2011).

35. Giddings, A. M. & Maitra, R. A Disease-Relevant High-Content Screening Assay to Identify Anti-Inflammatory Compounds for Use in Cystic Fibrosis. *J. Biomol. Screen.* **15**, 1204–1210 (2010).

36. Moffat, J. *et al.* A Lentiviral RNAi Library for Human and Mouse Genes Applied to an Arrayed Viral High-Content Screen. *Cell* **124**, 1283–1298 (2006).

37. Ouyang, W., Aristov, A., Lelek, M., Hao, X. & Zimmer, C. Deep learning massively accelerates super-resolution localization microscopy. *Nat. Biotechnol.* **36**, 460–468 (2018).

38. Keller, P. J. & Ahrens, M. B. Visualizing Whole-Brain Activity and Development at the Single-Cell Level Using Light-Sheet Microscopy. *Neuron* **85**, 462–483 (2015).

39. Zheng, Z. *et al.* A Complete Electron Microscopy Volume of the Brain of Adult Drosophila melanogaster. *Cell* **174**, 730-743.e22 (2018).

40. Borland, D. *et al.* Segmentor: a tool for manual refinement of 3D microscopy annotations. *BMC Bioinformatics* **22**, 260 (2021).

41. Femino, A. M., Fay, F. S., Fogarty, K. & Singer, R. H. Visualization of Single RNA Transcripts in Situ. *Science* (1998) doi:10.1126/science.280.5363.585.

42. Raj, A., van den Bogaard, P., Rifkin, S. A., van Oudenaarden, A. & Tyagi, S. Imaging individual mRNA molecules using multiple singly labeled probes. *Nat. Methods* **5**, 877–879 (2008).

43. Falconnet, D. *et al.* High-throughput tracking of single yeast cells in a microfluidic imaging matrix. *Lab. Chip* **11**, 466–473 (2011).

44. Buggenthin, F. *et al.* An automatic method for robust and fast cell detection in bright field images from high-throughput microscopy. *BMC Bioinformatics* **14**, 297 (2013).

45. Tinevez, J.-Y. *et al.* TrackMate: An open and extensible platform for single-particle tracking. *Methods* **115**, 80–90 (2017).

46. Amat, F. *et al.* Fast, accurate reconstruction of cell lineages from large-scale fluorescence microscopy data. *Nat. Methods* **11**, 951–958 (2014).

47. McDole, K. *et al.* In Toto Imaging and Reconstruction of Post-Implantation Mouse Development at the Single-Cell Level. *Cell* **175**, 859-876.e33 (2018).

48. Gustafsson, N. *et al.* Fast live-cell conventional fluorophore nanoscopy with ImageJ through super-resolution radial fluctuations. *Nat. Commun.* **7**, 12471 (2016).

49. Agard, D. A. & Sedat, J. W. Three-dimensional architecture of a polytene nucleus. *Nature* **302**, 676–681 (1983).

50. Sage, D. *et al.* DeconvolutionLab2: An open-source software for deconvolution microscopy. *Methods* **115**, 28–41 (2017).

51. Levet, F. *et al.* Developing open-source software for bioimage analysis: opportunities and challenges. *F1000Research* **10**, 302 (2021).

52. Schneider, C. A., Rasband, W. S. & Eliceiri, K. W. NIH Image to ImageJ: 25 years of image analysis. *Nat. Methods* **9**, 671–675 (2012).

53. Schindelin, J. *et al.* Fiji: an open-source platform for biological-image analysis. *Nat. Methods* **9**, 676–682 (2012).

54. Berg, S. *et al.* ilastik: interactive machine learning for (bio)image analysis. *Nat. Methods* **16**, 1226–1232 (2019).

55. Van Valen, D. A. *et al.* Deep Learning Automates the Quantitative Analysis of Individual Cells in Live-Cell Imaging Experiments. *PLOS Comput. Biol.* **12**, e1005177 (2016).

56. Kraus, O. Z., Ba, J. L. & Frey, B. J. Classifying and segmenting microscopy images with deep multiple instance learning. *Bioinformatics* **32**, i52–i59 (2016).

57. Singh, S., Carpenter, A. E. & Genovesio, A. Increasing the Content of High-Content Screening: An Overview. *J. Biomol. Screen.* **19**, 640–650 (2014).

58. Buades, A., Coll, B. & Morel, J. M. A Review of Image Denoising Algorithms, with a New One. *Multiscale Model. Simul.* **4**, 490–530 (2005).

59. Fan, L., Zhang, F., Fan, H. & Zhang, C. Brief review of image denoising techniques. *Vis. Comput. Ind. Biomed. Art* **2**, 7 (2019).

60. Whelan, D. R. & Bell, T. D. M. Image artifacts in Single Molecule Localization Microscopy: why optimization of sample preparation protocols matters. *Sci. Rep.* **5**, 7924 (2015).

61. Culley, S. *et al.* NanoJ-SQUIRREL: quantitative mapping and minimisation of super-resolution optical imaging artefacts. *Nat. Methods* **15**, 263–266 (2018).

62. Schaefer, L. H., Schuster, D. & Schaffer, J. Structured illumination microscopy: artefact analysis and reduction utilizing a parameter optimization approach. *J. Microsc.* **216**, 165–174 (2004).

63. Förster, R., Müller, W., Richter, R. & Heintzmann, R. Automated distinction of shearing and distortion artefacts in structured illumination microscopy. *Opt. Express* **26**, 20680–20694 (2018).

64. Dutta, A., Verma, Y. & Jawahar, C. V. Automatic image annotation: the quirks and what works. *Multimed. Tools Appl.* **77**, 31991–32011 (2018).

65. He, K., Zhang, X., Ren, S. & Sun, J. Deep Residual Learning for Image Recognition. in 770–778 (2016).

66. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436–444 (2015).

67. Pham, N.-Q., Kruszewski, G. & Boleda, G. Convolutional Neural Network Language Models. in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* 1153–1162 (Association for Computational Linguistics, 2016). doi:10.18653/v1/D16-1123.

68. Otter, D. W., Medina, J. R. & Kalita, J. K. A Survey of the Usages of Deep Learning for Natural Language Processing. *IEEE Trans. Neural Netw. Learn. Syst.* **32**, 604–624 (2021).

69. Rosenblatt, F. *The perceptron, a perceiving and recognizing automaton Project Para*. (Cornell Aeronautical Laboratory, 1957).

70. McCulloch, W. S. & Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* **5**, 115–133 (1943).

71. Albawi, S., Mohammed, T. A. & Al-Zawi, S. Understanding of a convolutional neural network. in *2017 International Conference on Engineering and Technology (ICET)* 1–6 (2017). doi:10.1109/ICEngTechnol.2017.8308186.

72. Moen, E. *et al. Accurate cell tracking and lineage construction in live-cell imaging experiments with deep learning*. http://biorxiv.org/lookup/doi/10.1101/803205 (2019) doi:10.1101/803205.

73. von Chamier, L., Laine, R. F. & Henriques, R. Artificial intelligence for microscopy: what you should know. *Biochem. Soc. Trans.* **47**, 1029–1040 (2019).

74. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning representations by back-propagating errors. *Nature* **323**, 533–536 (1986).

75. Bottou, L. Large-Scale Machine Learning with Stochastic Gradient Descent. in *Proceedings of COMPSTAT'2010* (eds. Lechevallier, Y. & Saporta, G.) 177–186 (Physica-Verlag HD, 2010). doi:10.1007/978-3-7908-2604-3_16.

76. Bengio, Y. Practical Recommendations for Gradient-Based Training of Deep Architectures. in *Neural Networks: Tricks of the Trade: Second Edition* (eds. Montavon, G., Orr, G. B. & Müller, K.-R.) 437–478 (Springer, 2012). doi:10.1007/978-3-642-35289-8_26.

77. Smith, L. N. Cyclical Learning Rates for Training Neural Networks. in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)* 464–472 (2017). doi:10.1109/WACV.2017.58.

78. Goodfellow, I., Bengio, Y. & Courville, A. *Deep learning*. (MIT press, 2016).

79. Bashir, D., Montañez, G. D., Sehra, S., Segura, P. S. & Lauw, J. An Information-Theoretic Perspective on Overfitting and Underfitting. in *AI 2020: Advances in Artificial Intelligence* (eds. Gallagher, M., Moustafa, N. & Lakshika, E.) 347–358 (Springer International Publishing, 2020). doi:10.1007/978-3-030-64984-5_27.

80. Belthangady, C. & Royer, L. A. Applications, promises, and pitfalls of deep learning for fluorescence image reconstruction. *Nat. Methods* **16**, 1215–1225 (2019).

81. Aytekin, C. Neural Networks are Decision Trees. Preprint at http://arxiv.org/abs/2210.05189 (2022).

82. Chen, X. *et al.* InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. *ArXiv160603657 Cs Stat* (2016).

83. Mullachery, V., Khera, A. & Husain, A. Bayesian Neural Networks. Preprint at https://doi.org/10.48550/arXiv.1801.07710 (2018).

84. Dalca, A. V. *et al.* Unsupervised Deep Learning for Bayesian Brain MRI Segmentation. in *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019* (eds. Shen, D. et al.) 356–365 (Springer International Publishing, 2019). doi:10.1007/978-3-030-32248-9_40.

85. Amini, A., Schwarting, W., Soleimany, A. & Rus, D. Deep Evidential Regression. in *Advances in Neural Information Processing Systems* vol. 33 14927–14937 (Curran Associates, Inc., 2020).

86. Test-time augmentation for deep learning-based cell segmentation on microscopy images | Scientific Reports. https://www.nature.com/articles/s41598-020-61808-3.

87. Aleatoric uncertainty estimation with test-time augmentation for medical image segmentation with convolutional neural networks - ScienceDirect. https://www.sciencedirect.com/science/article/pii/S0925231219301961.

88. Padilla, R., Netto, S. L. & da Silva, E. A. B. A Survey on Performance Metrics for Object-Detection Algorithms. in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)* 237–242 (IEEE, 2020). doi:10.1109/IWSSIP48289.2020.9145130.

89. Everingham, M. & Winn, J. The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Development Kit. 29.

90. Sun, C., Shrivastava, A., Singh, S. & Gupta, A. Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. in 843–852 (2017).

91. Möckl, L., Roy, A. R. & Moerner, W. E. Deep learning in single-molecule microscopy: fundamentals, caveats, and recent developments [Invited]. *Biomed. Opt. Express* **11**, 1633 (2020).

92. Everingham, M. *et al.* The Pascal Visual Object Classes Challenge: A Retrospective. *Int. J. Comput. Vis.* **111**, 98–136 (2015).

93. Girshick, R. Fast R-CNN. *ArXiv150408083 Cs* (2015).

94. Redmon, J. & Farhadi, A. YOLO9000: Better, Faster, Stronger. in 7263–7271 (2017).

95. Lin, T.-Y., Goyal, P., Girshick, R., He, K. & Dollár, P. Focal Loss for Dense Object Detection. *ArXiv170802002 Cs* (2018).

96. Taylor, L. & Nitschke, G. Improving Deep Learning with Generic Data Augmentation. in *2018 IEEE Symposium Series on Computational Intelligence (SSCI)* 1542–1547 (2018). doi:10.1109/SSCI.2018.8628742.

97. Shorten, C. & Khoshgoftaar, T. M. A survey on Image Data Augmentation for Deep Learning. *J. Big Data* **6**, 60 (2019).

98. Ferreira, C. A. *et al.* Classification of Breast Cancer Histology Images Through Transfer Learning Using a Pre-trained Inception Resnet V2. in *Image Analysis and Recognition* (eds. Campilho, A., Karray, F. & ter Haar Romeny, B.) 763–770 (Springer International Publishing, 2018). doi:10.1007/978-3-319-93000-8_86.

99. Demir, A., Yilmaz, F. & Kose, O. Early detection of skin cancer using deep learning architectures: resnet-101 and inception-v3. in *2019 Medical Technologies Congress (TIPTEKNO)* 1–4 (2019). doi:10.1109/TIPTEKNO47231.2019.8972045.

100. Raman, S., Maskeliūnas, R. & Damaševičius, R. Markerless Dog Pose Recognition in the Wild Using ResNet Deep Learning Model. *Computers* **11**, 2 (2022).

101. Modarres, M. H. *et al.* Neural Network for Nanoscience Scanning Electron Microscope Image Recognition. *Sci. Rep.* **7**, 13282 (2017).

102. Bierman, A. *et al.* A High-Throughput Phenotyping System Using Machine Vision to Quantify Severity of Grapevine Powdery Mildew. *Plant Phenomics* **2019**, 1–13 (2019).

103. Bolukbasi, T., Chang, K.-W., Zou, J. Y., Saligrama, V. & Kalai, A. T. Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings. in *Advances in Neural Information Processing Systems* vol. 29 (Curran Associates, Inc., 2016).

104. Shankar, S. *et al.* No Classification without Representation: Assessing Geodiversity Issues in Open Data Sets for the Developing World. *ArXiv171108536 Stat* (2017).

105. Caicedo, J. C., McQuin, C., Goodman, A., Singh, S. & Carpenter, A. E. Weakly Supervised Learning of Single-Cell Feature Embeddings. in 9309–9318 (2018).

106. Carneiro, G., Peng, T., Bayer, C. & Navab, N. Automatic Quantification of Tumour Hypoxia From Multi-Modal Microscopy Images Using Weakly-Supervised Learning Methods. *IEEE Trans. Med. Imaging* **36**, 1405–1417 (2017).

107. Chen, S. *et al.* Weakly Supervised Deep Learning for Detecting and Counting Dead Cells in Microscopy Images. in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)* 1737–1743 (2019). doi:10.1109/ICMLA.2019.00282.

108. Kallenberg, M. *et al.* Unsupervised Deep Learning Applied to Breast Density Segmentation and Mammographic Risk Scoring. *IEEE Trans. Med. Imaging* **35**, 1322–1331 (2016).

109. Sari, C. T. & Gunduz-Demir, C. Unsupervised Feature Extraction via Deep Learning for Histopathological Classification of Colon Tissue Images. *IEEE Trans. Med. Imaging* **38**, 1139–1149 (2019).

110. Lecun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **86**, 2278–2324 (1998).

111. Ronneberger, O., Fischer, P. & Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (eds. Navab, N., Hornegger, J., Wells, W. M. & Frangi, A. F.) 234–241 (Springer International Publishing, 2015). doi:10.1007/978-3-319-24574-4_28.

112. Segmentation of neuronal structures in EM stacks challenge - ISBI 2012. *ImageJ Wiki* https://imagej.github.io/events/isbi-2012-segmentation-challenge.

113. Weigert, M. *et al.* Content-aware image restoration: pushing the limits of fluorescence microscopy. *Nat. Methods* **15**, 1090–1097 (2018).

114. Ounkomol, C., Seshamani, S., Maleckar, M. M., Collman, F. & Johnson, G. R. Label-free prediction of three-dimensional fluorescence images from transmitted-light microscopy. *Nat. Methods* **15**, 917–920 (2018).

115. Çiçek, Ö., Abdulkadir, A., Lienkamp, S. S., Brox, T. & Ronneberger, O. 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation. in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016* (eds. Ourselin, S., Joskowicz, L., Sabuncu, M. R., Unal, G. & Wells, W.) vol. 9901 424–432 (Springer International Publishing, 2016).

116. Cai, S. *et al.* Dense-UNet: a novel multiphoton in vivo cellular image segmentation model based on a convolutional neural network. *Quant. Imaging Med. Surg.* **10**, 1275–1285 (2020).

117. Goodfellow, I. *et al.* Generative Adversarial Nets. in *Advances in Neural Information Processing Systems 27* (eds. Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D. & Weinberger, K. Q.) 2672–2680 (Curran Associates, Inc., 2014).

118. Zhu, J.-Y., Park, T., Isola, P. & Efros, A. A. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *ArXiv170310593 Cs* (2018).

119. Gilbert, E. N. & Pollak, H. O. Amplitude distribution of shot noise. *Bell Syst. Tech. J.* **39**, 333–350 (1960).

120. Sim, K. S., Thong, J. T. L. & Phang, J. C. H. Effect of shot noise and secondary emission noise in scanning electron microscope images. *Scanning* **26**, 36–40 (2004).

121. Snyder, D. L., Helstrom, C. W., Lanterman, A. D., Faisal, M. & White, R. L. Compensation for readout noise in CCD images. *JOSA A* **12**, 272–283 (1995).

122. Wilson, T. Resolution and optical sectioning in the confocal microscope. *J. Microsc.* **244**, 113–121 (2011).

123. Billinton, N. & Knight, A. W. Seeing the Wood through the Trees: A Review of Techniques for Distinguishing Green Fluorescent Protein from Endogenous Autofluorescence. *Anal. Biochem.* **291**, 175–197 (2001).

124. Periasamy, A. Fluorescence resonance energy transfer microscopy: a mini review. *J. Biomed. Opt.* **6**, 287–291 (2001).

125. Maddipatla, R. & Tankam, P. Bleed-through elimination method in a dual-channel fluorescence microscopy system. in *Multiphoton Microscopy in the Biomedical Sciences XX* vol. 11244 96–100 (SPIE, 2020).

126. Lehtinen, J. *et al.* Noise2Noise: Learning Image Restoration without Clean Data. in *Proceedings of the 35th International Conference on Machine Learning* 2965–2974 (PMLR, 2018).

127. Batson, J. & Royer, L. Noise2Self: Blind Denoising by Self-Supervision. in *Proceedings of the 36th International Conference on Machine Learning* 524–533 (PMLR, 2019).

128. Krull, A., Buchholz, T.-O. & Jug, F. Noise2Void - Learning Denoising from Single Noisy Images. *ArXiv181110980 Cs* (2019).

129. Wang, H. *et al.* Deep learning enables cross-modality super-resolution in fluorescence microscopy. *Nat. Methods* **16**, 103–110 (2019).

130. Nelson, A. J. & Hess, S. T. Molecular imaging with neural training of identification algorithm (neural network localization identification). *Microsc. Res. Tech.* **81**, 966–972 (2018).

131. Nehme, E., Weiss, L. E., Michaeli, T. & Shechtman, Y. Deep-STORM: super-resolution single-molecule microscopy by deep learning. *Optica* **5**, 458 (2018).

132. Christiansen, E. M. *et al.* In Silico Labeling: Predicting Fluorescent Labels in Unlabeled Images. *Cell* **173**, 792-803.e19 (2018).

133. Lu, A. X., Kraus, O. Z., Cooper, S. & Moses, A. M. Learning unsupervised feature representations for single cell microscopy images with paired cell inpainting. *PLOS Comput. Biol.* **15**, e1007348 (2019).

134. von Chamier, L. *et al.* Democratising deep learning for microscopy with ZeroCostDL4Mic. *Nat. Commun.* **12**, 2276 (2021).

135. Rivenson, Y. *et al.* Virtual histological staining of unlabelled tissue-autofluorescence images via deep learning. *Nat. Biomed. Eng.* **3**, 466–477 (2019).

136. Carpenter, A. E. *et al.* CellProfiler: image analysis software for identifying and quantifying cell phenotypes. *Genome Biol.* **7**, R100 (2006).

137. Sommer, C., Straehle, C., Köthe, U. & Hamprecht, F. A. Ilastik: Interactive learning and segmentation toolkit. in *2011 IEEE International Symposium on Biomedical Imaging: From Nano to Macro* 230–233 (2011). doi:10.1109/ISBI.2011.5872394.

138. Arganda-Carreras, I. *et al.* Trainable Weka Segmentation: a machine learning tool for microscopy pixel classification. *Bioinformatics* **33**, 2424–2426 (2017).

139. Ciresan, D., Gambardella, L. M., Giusti, A. & Schmidhuber, J. *Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images. Nips*. (2012).

140. Ning, F. *et al.* Toward automatic phenotyping of developing embryos from videos. *IEEE Trans. Image Process.* **14**, 1360–1371 (2005).

141. Kainz, P., Pfeiffer, M. & Urschler, M. Segmentation and classification of colon glands with deep convolutional neural networks and total variation regularization. *PeerJ* **5**, e3874 (2017).

142. BenTaieb, A. & Hamarneh, G. Topology Aware Fully Convolutional Networks for Histology Gland Segmentation. in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016* (eds. Ourselin, S., Joskowicz, L., Sabuncu, M. R., Unal, G. & Wells, W.) 460–468 (Springer International Publishing, 2016). doi:10.1007/978-3-319-46723-8_53.

143. Li, W. *et al.* Gland segmentation in colon histology images using hand-crafted features and convolutional neural networks. in *2016 IEEE 13th International Symposium on Biomedical Imaging (ISBI)* 1405–1408 (2016). doi:10.1109/ISBI.2016.7493530.

144. Litjens, G. *et al.* Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis. *Sci. Rep.* **6**, 26286 (2016).

145. Xu, J., Luo, X., Wang, G., Gilmore, H. & Madabhushi, A. A Deep Convolutional Neural Network for segmenting and classifying epithelial and stromal regions in histopathological images. *Neurocomputing* **191**, 214–223 (2016).

146. Litjens, G. *et al.* A survey on deep learning in medical image analysis. *Med. Image Anal.* **42**, 60–88 (2017).

147. Weigert, M., Schmidt, U., Haase, R., Sugawara, K. & Myers, G. Star-convex Polyhedra for 3D Object Detection and Segmentation in Microscopy. 8 (2020).

148. Schmidt, U., Weigert, M., Broaddus, C. & Myers, G. Cell Detection with Star-Convex Polygons. in *Medical Image Computing and Computer Assisted Intervention - MICCAI 2018 - 21st International Conference, Granada, Spain, September 16-20, 2018, Proceedings, Part II* 265–273 (2018). doi:10.1007/978-3-030-00934-2_30.

149. Mandal, S. & Uhlmann, V. Splinedist: Automated Cell Segmentation With Spline Curves. in *2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI)* 1082–1086 (2021). doi:10.1109/ISBI48211.2021.9433928.

150. Deng, J. *et al.* ImageNet: A large-scale hierarchical image database. in *2009 IEEE Conference on Computer Vision and Pattern Recognition* 248–255 (2009). doi:10.1109/CVPR.2009.5206848.

151. Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J. & Zisserman, A. The Pascal Visual Object Classes (VOC) Challenge. *Int. J. Comput. Vis.* **88**, 303–338 (2010).

152. Acevedo, A. *et al.* A dataset of microscopic peripheral blood cell images for development of automatic recognition systems. *Data Brief* **30**, 105474 (2020).

153. Simonyan, K. & Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv14091556 Cs* (2015).

154. Szegedy, C. *et al.* Going deeper with convolutions. in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 1–9 (2015). doi:10.1109/CVPR.2015.7298594.

155. Malon, C. D. & Cosatto, E. Classification of mitotic figures with convolutional neural networks and seeded blob features. *J. Pathol. Inform.* **4**, 9 (2013).

156. Xu, Z. & Huang, J. Detecting 10,000 Cells in One Second. in 676–684 (2016). doi:10.1007/978-3-319-46723-8_78.

157. Cireşan, D. C., Giusti, A., Gambardella, L. M. & Schmidhuber, J. Mitosis Detection in Breast Cancer Histology Images with Deep Neural Networks. in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2013* (eds. Mori, K., Sakuma, I., Sato, Y., Barillot, C. & Navab, N.) 411–418 (Springer, 2013). doi:10.1007/978-3-642-40763-5_51.

158. Shkolyar, A., Gefen, A., Benayahu, D. & Greenspan, H. Automatic detection of cell divisions (mitosis) in live-imaging microscopy images using Convolutional Neural Networks. in *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)* 743–746 (2015). doi:10.1109/EMBC.2015.7318469.

159. Pärnamaa, T. & Parts, L. Accurate Classification of Protein Subcellular Localization from High-Throughput Microscopy Images Using Deep Learning. *G3 GenesGenomesGenetics* **7**, 1385–1392 (2017).

160. Godinez, W. J., Hossain, I., Lazic, S. E., Davies, J. W. & Zhang, X. A multi-scale convolutional neural network for phenotyping high-content cellular images. *Bioinformatics* **33**, 2010–2019 (2017).

161. Richmond, D., Jost, A., Lambert, T., Waters, J. & Elliott, H. DeadNet: Identifying Phototoxicity from Label-free Microscopy Images of Cells using Deep ConvNets. (2017).

162. Eulenberg, P. *et al.* Reconstructing cell cycle and disease progression using deep learning. *Nat. Commun.* **8**, 463 (2017).

163. Kusumoto, D. *et al.* Automated Deep Learning-Based System to Identify Endothelial Cells Derived from Induced Pluripotent Stem Cells. *Stem Cell Rep.* **10**, 1687–1695 (2018).

164. Tan, M. & Le, Q. V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. Preprint at http://arxiv.org/abs/1905.11946 (2020).

165. Papers with Code - Detectron2. https://paperswithcode.com/lib/detectron2.

166. He, K., Gkioxari, G., Dollar, P. & Girshick, R. Mask R-CNN. in 2961–2969 (2017).

167. Kwok, S. Multiclass Classification of Breast Cancer in Whole-Slide Images. in *Image Analysis and Recognition* (eds. Campilho, A., Karray, F. & ter Haar Romeny, B.) 931–940 (Springer International Publishing, 2018). doi:10.1007/978-3-319-93000-8_106.

168. Al-Haija, Q. A. & Adebanjo, A. Breast Cancer Diagnosis in Histopathological Images Using ResNet-50 Convolutional Neural Network. in *2020 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)* 1–7 (2020). doi:10.1109/IEMTRONICS51293.2020.9216455.

169. The importance of technological advances. *Nat. Cell Biol.* **2**, E37–E37 (2000).

170. Latour, B. & Woolgar, S. *Laboratory life*. (Princeton University Press, 2013).

171. Martens, K. J. A. *et al.* Visualisation of dCas9 target search in vivo using an open-microscopy framework. *Nat. Commun.* **10**, 3552 (2019).

172. Ma, H., Fu, R., Xu, J. & Liu, Y. A simple and cost-effective setup for super-resolution localization microscopy. *Sci. Rep.* **7**, 1542 (2017).

173. Rogers, E. M. *Diffusion of innovations*. (Simon and Schuster, 2010).

174. Harris, C. R. *et al.* Array programming with NumPy. *Nature* **585**, 357–362 (2020).

175. Virtanen, P. *et al.* SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nat. Methods* **17**, 261–272 (2020).

176. Falk, T. *et al.* U-Net: deep learning for cell counting, detection, and morphometry. *Nat. Methods* **16**, 67–70 (2019).

177. Haberl, M. G. *et al.* CDeep3M—Plug-and-Play cloud-based deep learning for image segmentation. *Nat. Methods* **15**, 677–680 (2018).

178. Bannon, D. *et al. DeepCell Kiosk: Scaling deep learning-enabled cellular image analysis with Kubernetes*. http://biorxiv.org/lookup/doi/10.1101/505032 (2018) doi:10.1101/505032.

179. Belevich, I. & Jokitalo, E. *DeepMIB: User-friendly and open-source software for training of deep learning network for biological image segmentation*. http://biorxiv.org/lookup/doi/10.1101/2020.07.13.200105 (2020) doi:10.1101/2020.07.13.200105.

180. Hollandi, R., Szkalisity, A. & Toth, T. nucleAIzer: A Parameter-free Deep Learning Framework for Nucleus Segmentation Using Image Style Transfer. *Cell Syst.* **10**, 453-458.E6 (2020).

181. Moehl, C. & Dhole, P. Yapic. https://yapic.github.io/yapic/ (2020).

182. Ouyang, W., Mueller, F., Hjelmare, M., Lundberg, E. & Zimmer, C. ImJoy: an open-source computational platform for the deep learning era. *Nat. Methods* **16**, 1199–1200 (2019).

183. McQuin, C. *et al.* CellProfiler 3.0: Next-generation image processing for biology. *PLOS Biol.* **16**, e2005970 (2018).

184. Buchholz, T.-O., Prakash, M., Schmidt, D., Krull, A. & Jug, F. DenoiSeg: Joint Denoising and Segmentation. in *Computer Vision – ECCV 2020 Workshops* (eds. Bartoli, A. & Fusiello, A.) 324–337 (Springer International Publishing, 2020). doi:10.1007/978-3-030-66415-2_21.

185. Castelvecchi, D. Can we open the black box of AI? *Nat. News* **538**, 20 (2016).

186. Kluyver, T. *et al.* Jupyter Notebooks – a publishing format for reproducible computational workflows. in (eds. Loizides, F. & Scmidt, B.) 87–90 (IOS Press, 2016). doi:10.3233/978-1-61499-649-1-87.

187. Schmidt, U., Weigert, M., Broaddus, C. & Myers, G. Cell Detection with Star-Convex Polygons. in *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018* (eds. Frangi, A. F., Schnabel, J. A., Davatzikos, C.,

Alberola-López, C. & Fichtinger, G.) vol. 11071 265–273 (Springer International Publishing, 2018).

188. Salehi, P. & Chalechale, A. Pix2Pix-based Stain-to-Stain Translation: A Solution for Robust Stain Normalization in Histopathology Images Analysis. in *2020 International Conference on Machine Vision and Image Processing (MVIP)* 1–7 (2020). doi:10.1109/MVIP49855.2020.9116895.

189. Zhang, D. *et al.* Nuclei instance segmentation with dual contour-enhanced adversarial network. in *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)* 409–412 (2018). doi:10.1109/ISBI.2018.8363604.

190. Paszke, A. *et al.* PyTorch: An Imperative Style, High-Performance Deep Learning Library. in *Advances in Neural Information Processing Systems 32* (eds. Wallach, H. et al.) 8024–8035 (Curran Associates, Inc., 2019).

191. Abadi, M. *et al.* TensorFlow: A System for Large-Scale Machine Learning | USENIX. in (2016).

192. scikit-image ssim package.

193. Wang, Z., Bovik, A. C., Sheikh, H. R. & Simoncelli, E. P. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Trans. Image Process.* **13**, 600–612 (2004).

194. Zhou Wang & Bovik, A. C. Mean squared error: Love it or leave it? A new look at Signal Fidelity Measures. *IEEE Signal Process. Mag.* **26**, 98–117 (2009).

195. Jaccard, P. THE DISTRIBUTION OF THE FLORA IN THE ALPINE ZONE.1. *New Phytol.* (1912) doi:10.1111/j.1469-8137.1912.tb05611.x.

196. Bloice, M. D., Roth, P. M. & Holzinger, A. Biomedical image augmentation using Augmentor. *Bioinformatics* **35**, 4522–4524 (2019).

197. Jacquemet, G. ZeroCostDL4Mic - CARE (2D) example training and test dataset. (2020) doi:10.5281/ZENODO.3713330.

198. Jacquemet, G. & Chamier, L. V. ZeroCostDL4Mic - YoloV2 example training and test dataset. (2020) doi:10.5281/ZENODO.3941908.

199. Jukkala, J. & Jacquemet, G. ZeroCostDL4Mic - Stardist example training and test dataset. (2020) doi:10.5281/ZENODO.3715492.

200. Jacquemet, G. ZeroCostDL4Mic - CARE (3D) example training and test dataset. (2020) doi:10.5281/ZENODO.3713337.

201. Spahn, C. & Heilemann, M. ZeroCostDL4Mic - Label-free prediction (fnet) example training and test dataset. (2020) doi:10.5281/ZENODO.3748967.

202. Leterrier, C. & Laine, R. F. ZeroCostDL4Mic - DeepSTORM training and example dataset. (2020) doi:10.5281/ZENODO.3959089.

203. Jacquemet, G. ZeroCostDL4Mic - Noise2Void (3D) example training and test dataset. (2020) doi:10.5281/ZENODO.3713326.

204. Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 779–788 (IEEE, 2016). doi:10.1109/CVPR.2016.91.

205. Kirillov, A., He, K., Girshick, R., Rother, C. & Dollár, P. Panoptic Segmentation. *ArXiv180100868 Cs* (2019).

206. Zhang, R., Isola, P., Efros, A. A., Shechtman, E. & Wang, O. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. Preprint at http://arxiv.org/abs/1801.03924 (2018).

207. Oheim, M. & Li, D. Quantitative Colocalisation Imaging: Concepts, Measurements, and Pitfalls. in *Imaging Cellular and Molecular Biological Functions* (eds. Shorte, S. L. & Frischknecht, F.) 117–155 (Springer, 2007). doi:10.1007/978-3-540-71331-9_5.

208. Hollandi, R. *et al.* Nucleus segmentation: towards automated solutions. *Trends Cell Biol.* **32**, 295–310 (2022).

209. Zimmermann, T. Spectral Imaging and Linear Unmixing in Light Microscopy. in *Microscopy Techniques* (ed. Rietdorf, J.) vol. 95 245–265 (Springer Berlin Heidelberg, 2005).

210. McRae, T. D., Oleksyn, D., Miller, J. & Gao, Y.-R. Robust blind spectral unmixing for fluorescence microscopy using unsupervised learning. *PLOS ONE* **14**, e0225410 (2019).

211. Gammon, S. T., Leevy, W. M., Gross, S., Gokel, G. W. & Piwnica-Worms, D. Spectral unmixing of multicolored bioluminescence emitted from heterogeneous biological sources. *Anal. Chem.* **78**, 1520–1527 (2006).

212. Spectral Unmixing Plugins. https://imagej.nih.gov/ij/plugins/spectral-unmixing.html.

213. Prangemeier, T., Wildner, C., Françani, A. O., Reich, C. & Koeppl, H. Multiclass Yeast Segmentation in Microstructured Environments with Deep Learning. in *2020 IEEE Conference on Computational Intelligence in*

Bioinformatics and Computational Biology (CIBCB) 1–8 (2020). doi:10.1109/CIBCB48159.2020.9277693.

214. Guerrero-Peña, F. A. *et al.* Multiclass Weighted Loss for Instance Segmentation of Cluttered Cells. in *2018 25th IEEE International Conference on Image Processing (ICIP)* 2451–2455 (2018). doi:10.1109/ICIP.2018.8451187.
215. Rueden, C. T. *et al.* ImageJ2: ImageJ for the next generation of scientific image data. *BMC Bioinformatics* **18**, 529 (2017).
216. Gómez-de-Mariscal, E. *et al. DeepImageJ: A user-friendly plugin to run deep learning models in ImageJ*. http://biorxiv.org/lookup/doi/10.1101/799270 (2019) doi:10.1101/799270.
217. Isola, P., Zhu, J.-Y., Zhou, T. & Efros, A. A. Image-to-Image Translation with Conditional Adversarial Networks. *ArXiv161107004 Cs* (2018).
218. Zhu, J.-Y., Park, T., Isola, P. & Efros, A. A. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *ArXiv170310593 Cs* (2018).
219. Spahn, C. *et al.* DeepBacs: Bacterial image analysis using open-source deep learning approaches. 2021.11.03.467152 Preprint at https://doi.org/10.1101/2021.11.03.467152 (2021).
220. Krull, A., Buchholz, T.-O. & Jug, F. Noise2Void - Learning Denoising from Single Noisy Images. *ArXiv181110980 Cs* (2019).
221. Ouyang, W., Le, T., Xu, H. & Lundberg, E. Interactive biomedical segmentation tool powered by deep learning and ImJoy. Preprint at https://doi.org/10.12688/f1000research.50798.1 (2021).
222. Sofroniew, N. *et al.* napari/napari: 0.4.15. (2022) doi:10.5281/ZENODO.3555620.